UNIVERSIDAD AUTONÓMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Diseño e implementación de un sistema de representación de flujos de red

Autor: Raúl París Murillo
Tutor: Jorge Enrique López de Vergara Méndez

JULIO 2019

Raúl París Murillo
Diseño e implementación de un sistema de representación de flujos de red

Raúl París Murillo
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia, profesores y amigos

La simplicidad es la máxima sofisticación

Leonardo Da Vinci

Diseño e implementación de un sistema de representación de flujos de red

AUTOR: Raúl París Murillo
TUTOR: Jorge Enrique López de Vergara Méndez

High Performance Computing and Networking Research Group (HPCN)
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2019

Resumen

A lo largo de las últimas cinco décadas, Internet se ha ido convirtiendo en la herramienta fundamental que es en la actualidad, siendo empleada tanto de forma personal como empresarial. Esta evolución de Internet ha supuesto un gran aumento de usuarios, y, por ende, un incremento masivo en la cantidad de tráfico que circula por la red. De esta forma, y debido a la cantidad masiva de usuario que posee en la actualidad, ha generado la necesidad de monitorizar el tráfico que circula a través de Internet, para su posterior análisis y almacenamiento de registros.

Anteriormente, los sistemas de monitorización de flujos de red empleaban mayormente todos los campos de cada paquete, incluyendo el contenido, para mostrar la información contenida en los paquetes que circulaban por la red. En cambio, esta forma de representación se ha quedado obsoleta con el tiempo. La causa de su obsolescencia es la imposibilidad de la utilización de paquetes, puesto a todo el espacio que ocupan, pasando a utilizar flujos para la representación masiva de datos. Para este análisis ya no se necesita una gran cantidad de información de cada paquete procesado, sino, solo la información que el usuario considere necesaria para el análisis de los datos de la red en conjunto. Esta gestión de la red se lleva a cabo en cuadros de mando, dónde se seleccionan los datos a representar, el tipo de gráfica mostrada y los filtros que se desean realizar.

En este Trabajo Fin de Grado, se plantea diseñar y desarrollar un sistema basado en técnicas de tratamiento masivo de datos que permita realizar una monitorización, y su posterior análisis de una forma visual. Para implementar el sistema deseado, se ha analizado las distintas herramientas que existen en el mercado actual, y, posteriormente se ha seleccionado las herramientas finalmente utilizadas en el desarrollo, utilizando la plataforma de visualización Grafana.

Ante la ausencia de un enlace entre la plataforma de visualización y el colector de flujos, se ha desarrollado e implementado un servidor HTTP construido sobre el lenguaje de programación Python. Por último, se han realizado pruebas para evaluar el rendimiento del sistema diseñado frente al sistema más popular en la actualidad, la pila ELK (Elasticsearch, Logstash, Kibana).

Después de realizar pruebas en ambos sistemas, se ha demostrado que la pila ELK tiene un mejor rendimiento que el sistema desarrollado en lo que se refiere a tiempo de búsqueda. Sin embargo, el sistema desarrollado ha demostrado ser mucho mejor en las pruebas de almacenamiento de los registros.

Palabras Clave

Monitorización de tráfico, base de datos masiva, tratamiento masivo de datos, servidor HTTP, fuente de datos, cuadro de mando, Grafana, YAF, SILK, flujo de red.

Abstract

Along the past five decades, the Internet has become a key tool that is used daily for a variety of purposes, spanning from personal ones to business ones. The Internet's evolution has seen a noticeable increase in the number of users, which in turn has caused an increase of the Internet's traffic. Moreover, due to the massive quantity of Internet users that can be seen nowadays, there is an impeding need for controlling the Internet's traffic, in order to analyse and store it.

Previously, the netflow monitoring systems mainly showed the information contained within each of the packages that went through the net. However, such a representation technique had become outdated. The main cause of such an obsolescence is the impossibility of using packets, because all the disk space the need, switching to use flows to represent massive data visualisations, whereas now there is no such need of having a gigantic quantity of information for each of the processed packages. The only thing that is needed is the overall information which is being used to analyse along with the rest of the net data. This management is done using dashboards, where the user selects the represented data, the type of representation and the filters which will be applied to the data shown.

This end-of-degree thesis will therefore present the design and development of a Big Data-based system which enables the monitoring and the subsequent visual analysis of netflow data. In order to implement such a system, different tools available from the market have been analysed and, subsequently, chosen for the development of the system, using the visual platform Grafana for the representations.

Due to the lack of a direct link between the visualisation platform and the netflow collector, a Python-based HTTP server has been developed and implemented. Finally, the designed system has been tested to evaluate its performance compared to that of the most popular system nowadays, the ELK STACK (Elasticsearch, Logstash, Kibana).

After testing both systems, it has been proven that the ELK STACK system showcases a better search time performance. Nonetheless, the designed system proved to be much better during the register storage tests.

Key Words

Traffic monitoring, massive database, Big Data, HTTP server, data source, dashboard, Grafana, YAF, SILK, netflow.

AGRADECIMIENTOS

Me parece mentira estar ya redactando el apartado de agradecimientos del TFG. ¡Qué rápido pasa el tiempo! Hoy termina esta bonita etapa de mi vida, donde he conocido a grandes amigos y muy buenos profesores, que espero que me acompañen a lo largo de mi vida. En especial me gustaría agradecer el apoyo de toda mi familia sin excepción, abuelos, tíos, primos y cuñada. A mi abuela Quica, por todas sus novenas a San Judas Tadeo para que aprobara en los exámenes difíciles, que parece, que han dado resultado.

Pero quiero dedicar un espacio a parte para mi familia más cercana, a los que nunca les agradeceré suficiente todo su cariño y todo lo que han hecho por mí.

Mi padre, que con su tenacidad me ha enseñado lo que significa el trabajo duro y la constancia. Mi madre, que día tras día me ha enseñado a tener todo bajo control y a manejar situaciones difíciles. Mi hermano, que, con su ejemplo y su apoyo, me ha guiado en la vida, enseñándome todo lo que se necesita para ser un buen ingeniero y una increíble persona, cómo es él.

También quiero nombrar a Victoria, mi novia. Ella, con su paciencia y amor, me ha enseñado que la ambición, si la juntas con esfuerzo y constancia, te hace lograr cualquier cosa que te propongas. Me ha apoyado en todo este trayecto, y debo agradecer su infinita comprensión en todas las tardes que se quedaba junto a mí, porque yo necesitaba estudiar.

Por último, pero no menos importante, me gustaría nombrar a mi tutor, Jorge Enrique López de Vergara Méndez. Él me ha transmitido toda su sabiduría con todo el esfuerzo que ha puesto día a día en el proyecto, ha tenido una paciencia titánica esperando los avances y ha estado pendiente de mi para todo lo que he necesitado para lograr este gran objetivo, siempre con amabilidad y respeto.

Aprendiz de todo y maestro de nada.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Glosario

ISP	*Internet Service Provider*	Proveedor de servicios de Internet
VOD	*Video on Demand*	Video bajo demanda
VoIP	*Voice over IP*	Voz sobre IP
IoT	*Internet of Things*	Internet de las cosas
IPFIX	*IP Flow Information Export*	Protocolo de internet sobre la información de exportación de flujos
ELK	Elasticseach Logstash Kibana	
PCAP	*Packet Capture*	Captura de paquetes
JSON	*JavaScript Object Notation*	Notación de objetos Javascript
CSV	*Comma-Separated Values*	Separación de valores por comas
PCAPNG	*Packet Capture Next Generation*	Captura de paquetes de siguiente generación
HTTP	*HyperText Transfer Protocol*	Protocolo de transferencia de hipertexto
IP	*Internet Protocol*	Protocolo de Internet
HTML	*HyperText Markup Language*	Leguaje marcado de hipertexto
TCP	*Transmission Control Protocol*	Protocolo de control de transmisión
GB	Gigabyte	
SQL	*Structured Query Language*	Lenguaje de petición estructurado

1 Introducción

1.1 Motivación

Desde su creación en el año 1969 bajo el término de ARPANET, Internet (popularmente conocida como "la *red*") se ha ido desarrollando y ha adquirido un tamaño titánico. Naciendo como un proyecto para conectar telemáticamente las universidades de Estados Unidos, ha ido ganando usuarios, y de esta forma, ha ido incrementando el tráfico a una velocidad desmesurada. Debido a este crecimiento, Internet se ha ido transformando en lo que es hoy, una herramienta clave en nuestra sociedad para el uso personal, e incluso para el desarrollo empresarial. Las estadísticas del crecimiento se pueden observar en la Tabla 1-1.

Tabla 1-1 Datos globales del uso de Internet [1].

USO GLOBAL DE INTERNET Y ESTADÍSTICAS DE LA POBLACIÓN MARZO, 2019 – NUEVA ACTUALIZACIÓN						
Regiones	Población (Mill)	Población % Del Mundo	Usuarios (Mill) de Internet	Tasa de Penetración %	Crecimiento 2000-2019	Usuarios Mundiales %
África	1.320,038	17,1%	492,762	37,3%	10.815%	11,2%
Asia	4.241,972	55,0%	2.197,444	51,8%	1.822%	50,1%
Europa	829,173	10,7%	719,365	86,8%	584%	16,4%
Latino América	658,345	8,5%	444,493	67,5%	2.360%	10,1%
Oriente Medio	258,356	3,3%	173,542	67,2%	5.183%	4,0%
Norte América	366,496	4,7%	327,568	89,4%	203%	7,5%
Oceanía	41,839	0,5%	28,634	68,4%	276%	0,7%
TOTAL	7.716,223	100%	4.383,810	56,8%	1.114%	100%

Basándonos en los datos proporcionados por *Internet World Stats* [1], podemos apreciar la revolución que ha vivido la sociedad en estas cinco décadas. Desde comienzos de siglo, en el año 2000, los usuarios de Internet han crecido en un 1.114%, y el porcentaje de la población que utiliza Internet supone 4 346 561 853 usuarios, una cifra estratosférica. Más de la mitad de los habitantes del mundo utilizan Internet en la actualidad, concretamente, el 56% de la población global.

Tal es la importancia de Internet actualmente, que las 4 empresas mejor valoradas en bolsa son tecnológicas, del ámbito de las TIC [2].Estas empresas basan casi la totalidad de su comunicación con los clientes a través de Internet, además lanzan sus productos anunciándolos en Internet y también envían decenas de actualizaciones a través de este medio. Casi la totalidad de sus acciones, son realizadas por Internet, llegando al punto de casi no poseer tiendas físicas o utilizarlas más como una herramienta de marketing. Esto permite a las empresas innovar e intentar encontrar la forma de mantener sus registros y almacenar la información de sus millones de clientes. En la Figura 1.1-1se puede observar los valores en bolsa de las cuatro empresas mejor valoradas.

Figura 1.1-1 Ranking de empresas por valor en bolsa [2].

El rápido incremento en el tráfico por la red también se debe en gran medida a la mejora de los servicios suministrados por los *Internet Service Provider* (ISPs, Proveedores de Servicio de Internet), que, a medida que se escalaban, proporcionaban mejores servicios, alcanzando la calidad actual, donde el usuario final tiene en su vivienda acceso a cientos de Megabits/segundo (MB/s). En consecuencia, los servicios como el correo electrónico, ha ido evolucionando de un estado donde sólo se podía enviar texto, hasta lo que conocemos hoy, un servicio en el que se puede enviar texto, más un sinfín de archivos adjuntos, tales como imágenes o videos.

En concordancia con esto también han surgido nuevos servicios que, en los inicios de Internet, no se habían imaginado, como, por ejemplo, el video bajo demanda (VOD, *Video on Demand*) o la voz sobre IP (VoIP, *Voice over IP*). En la actualidad el crecimiento del tráfico de datos se está incrementando año tras año con las diversas ramas que están surgiendo mediante el uso de Internet, una de ellas es la Internet de las Cosas *(IoT, Internet of Things),* que enlazaría dispositivos de uso cotidiano como pueden ser un frigorífico para automatizar el proceso de compra, o comunicarlo con nuestro *smartphone* para enviar una notificación cada vez que un producto esté a punto de agotarse. Como se puede intuir, todo este nuevo tráfico de datos aumentará en gran medida el tráfico actual.

Debido a todo lo anterior, surge la necesidad de trabajar con grandes volúmenes de datos, esto hoy en día es viable gracias al *Big Data*. Mediante las soluciones de *Big Data* se puede almacenar y gestionar de forma masiva repositorios de datos con altísima volumetría [3], y aplicando sobre estos técnicas de *Data Mining*, se es capaz de convertir estos datos en información útil. A lo largo de los últimos años ha ido creciendo el deseo de conocer y registrar el tráfico que recorre internet, almacenar este tráfico, y, lo más importante, ser capaces de filtrar y hacer un correcto *Data Mining* del mismo, para poder extraer de una abrumadora cantidad de flujos y paquetes, información que pueda ser utilizada, administrada y visualizada.

1.2 Objetivos

En concordancia con lo expuesto con anterioridad, el objetivo de este Trabajo Fin de Grado (TFG) es desarrollar un sistema de monitorización de red basado en técnicas de captura y tratamiento masivo de datos que posteriormente permitan ser filtrados y analizados de una manera visual. Con este sistema el usuario tendrá la capacidad de filtrar según los parámetros que decida, crear diferentes gráficas, figuras y además observar qué sucede en tiempo real. Para alcanzar el objetivo final, se ha decidido dividirlo en tres subobjetivos. Estos subobjetivos son los siguientes:

- **Captura del tráfico y almacenamiento**
  Es necesario capturar la totalidad del tráfico que atraviesa nuestra red, ya sea de salida o, de entrada. Todos los flujos que atraviesen la red deberán de ser capturados, para su posterior almacenamiento en archivos con formato IPFIX [4] que permiten ahorrar espacio frente a otros tipos de indexado como PCAP [5] o JSON [6].

- **Filtrado de los datos**
  Una vez almacenados los datos, cada vez que el usuario quiera filtrar algún campo en la representación, o cada vez que ponga los límites temporales a las gráficas necesitaremos realizar un filtrado de todos los paquetes de una forma óptima para acelerar este proceso.

- **Representación de los datos**
  Por último, la información filtrada deberá ser representada de forma gráfica para la explotación por parte del usuario final, utilizando para ello las plataformas de representación analítica de datos. En consecuencia, el usuario obtendrá una forma sencilla de visualizar el estado de la red. La comunicación entre la plataforma utilizada y nuestra base de datos será realizada con un servidor en Python [7].

1.3 Fases de realización del proyecto

El Trabajo Fin de Grado (TFG) en su inicio, comenzó con una fase de documentación, analizando bibliografía y documentación oficial de los programas a utilizar. Posteriormente, entramos en la fase de intentos de enlazar los diversos programas utilizados para una captura de tráfico óptima. Después pasamos a la etapa de creación del servidor Python. Y, por último, la fase de pruebas de todo el sistema enlazado, comparándolo frente a otros sistemas similares. Todas estas fases detalladas por semanas se pueden apreciar en el siguiente diagrama de Gantt.

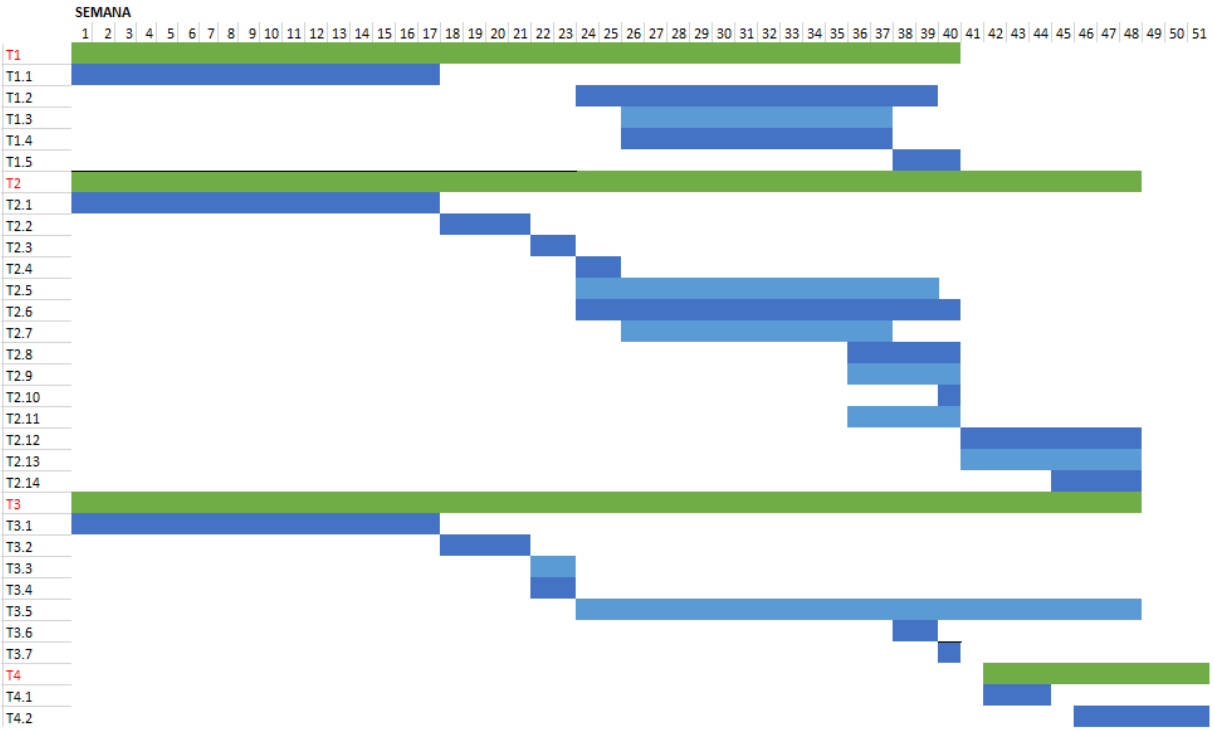

Figura 1.3-1 Diagrama de Gantt

T1	Implementación del colector de flujos
T1.1	Estudio inicial de las herramientas YAF y SILK
T1.2	Analizar la posibilidad de utilizar Pysilk (función de SILK) en Python 3
T1.3	Análisis para la creración de flujos unidireccionales en YAF
T1.4	Comparar el almacenamiento de SILK con Elasticsearch
T1.5	Intentar realizar los agrupamientos de paquetes con funciones de SILK
T2	Implementación del servidor HTTP en Python
T2.1	Estudio inicial de servidores HTTP Python
T2.2	Implementación de una primera versión del servidor HTTP
T2.3	Utilizar librería de Python para codificar y decodificar JSON
T2.4	Creación de un programa Python que extraíga campos de SILK
T2.5	Implementar funciones para agregar datos de flujos según el intervalo de tiempo
T2.6	Continuar implementando el servidor HTTP
T2.7	Poder definir filtro según la petición de Grafana
T2.8	Añadir try/excepts al servidor HTTP
T2.9	Comprobar si editar el JSON de SILK es más rápido que construir uno
T2.10	Revisar el zoom con el manejo de UTC frente a hora local
T2.11	Realizar prueba de espacio de almacenamiento
T2.12	Realizar prueba de velocidad
T2.13	Diferenciar paneles de Grafana
T2.14	Intentar realizar una prueba con datos reales
T3	Implementación de Grafana
T3.1	Estudio inicial de fuentes de datos en Grafana
T3.2	Observar la estructura de las peticiones de Grafana
T3.3	Observar algún ejemplo de fuentes de datos de Grafana
T3.4	Cargar flujos desde Elasticsearch a Grafana
T3.5	Crear cuadro de mandos basado en ejemplos
T3.6	Añadir variables en Grafana
T3.7	Añadir filtros de destino IP, puerto, campo abierto Pysilk
T4	Memoria
T4.1	Creación de índice de la memoria
T4.2	Realización de la memoria

Figura 1.3-2 Fases del proyecto

1.4 Organización de la memoria

Para que los resultados extraídos del trabajo sean fácilmente comprensibles, se va a exponer y enumerar de una forma clara todas las fases del mismo. A lo largo de la memoria se observará la evolución y los pasos seguidos para su entera realización. El resto de esta memoria consta de los siguientes capítulos:

- **Capítulo 2: Estado del arte.** En este capítulo se presentarán las principales tecnologías y los conceptos básicos para entender el trabajo realizado. Las tecnologías expuestas serán las que existen actualmente en los campos de captura de tráfico, de almacenamiento y de filtrado masivo de datos.
- **Capítulo 3: Diseño.** En este capítulo se presentará la hipótesis inicial planteada, y la implementación final desarrollada. En la hipótesis inicial se observará las elecciones iniciales de los programas y se expondrá el criterio por el cual han sido elegidos los programas y las plataformas empleadas finalmente.
- **Capítulo 4: Desarrollo.** En este capítulo se presentará la evolución sobre la hipótesis inicial. Las decisiones que se han ido tomando para alcanzar los objetivos y la implementación que se ha realizado para lograr un funcionamiento de forma óptima. Así mismo, se observará cómo se plantea el funcionamiento del sistema para diversos escenarios.
- **Capítulo 5: Integración, pruebas y resultados.** En este capítulo se presentarán los resultados de las pruebas a las que se ha sometido el sistema desarrollado en este TFG. También se compararán los resultados obtenidos, con los de los principales sistemas existentes en la actualidad, que realizan los mismos procesos.
- **Capítulo 6: Conclusiones y trabajo futuro.** En última instancia, en este capítulo se presentarán las conclusiones extraídas de la realización de este Trabajo Fin de Grado, la capacidad real del sistema desarrollado y las posibles investigaciones que se podrán desarrollar a partir de la realización de este Trabajo Fin de Grado.

2 Estado del arte

En este capítulo se presentarán los conceptos básicos y las principales tecnologías para entender el trabajo realizado. Las tecnologías empleadas y utilizadas serán las que existen actualmente en los campos de captura de tráfico, de almacenamiento y de filtrado masivo de datos. De esta forma, se hará una introducción en la visión actual sobre esta área de la información.

2.1 Colectores de flujo

Un flujo es un resumen de un paquete recibido de la red en el que se almacenan las características del paquete, pero no el contenido del mismo. Un exportador de flujo es un programa o herramienta que captura el tráfico que circula por nuestro dispositivo, nuestra red privada o una red a la que se tenga acceso. Un colector de flujo tiene la función de recibir los datos enviados desde el exportador y guardar el tráfico capturado en el formato deseado por el usuario. Dependiendo de la herramienta utilizada, la transformación o almacenamiento tendrá más o menos posibilidades. Los principales formatos en los que se suelen almacenar las capturas de tráfico son JSON, CSV [8] e IPFIX, de los que hablaremos más adelante.

En la actualidad las tecnologías de los colectores de flujo están muy desarrolladas, otorgando un abanico bastante amplio al usuario, desde distintas interfaces gráficas, hasta la posibilidad de establecer alertas ante posibles paquetes maliciosos o un aumento repentino del tráfico de la red, como es el caso del sistema de Solarwinds [9]. Día a día se mejora la capacidad de los colectores de flujo, haciéndolos más eficientes al procesar datos, incrementando las funciones disponibles y facilitando el interfaz para el usuario. Pero al crear un colector de flujo hay que tomar la decisión crucial, de qué formato será el utilizado para almacenar los datos.

Esta diferencia aparentemente insignificante puede suponer que una herramienta quede obsoleta, o por contra, comience a diferenciarse de sus competidores. Como ejemplo, se puede nombrar a *Netflow collector* (Colector de flujo de red) de Solarwinds, una herramienta archiconocida actualmente. Esta herramienta tiene por defecto SQL como forma de almacenar los archivos. SQL (*Structured Query Language*, Lenguaje de petición estructurado) es un formato para estructurar datos, este formato es muy común en el ámbito de gestiones de datos.

El analizador de tráfico de red de Solarwinds es una potente herramienta de análisis de tráfico de red. Esta herramienta tiene un amplio catálogo de funciones disponibles, dependiendo de lo que el usuario esté dispuesto a pagar. Estas funciones van desde el análisis de la red mostrándose por pantalla en un tablero, el monitoreo del ancho de banda, las alertas de picos de tráfico en la red, la identificación de flujos maliciosos o mal formados, hasta el monitoreo del ancho de banda en conexiones inalámbricas con aplicaciones.

El analizador de tráfico de datos captura los datos de la red que se seleccione, verifica que no se trata de un flujo malicioso o malformado, posteriormente indexa los datos en la base de datos SQL. Una vez almacenados, realiza análisis sobre ellos por si es necesaria la activación de alguna alerta en el interfaz del usuario y después muestra las estadísticas del tráfico de la red actualizadas en los tableros disponibles.

Almacenar los flujos de la red en formato SQL hace que el tiempo desde que se realiza una petición por parte del usuario para mostrar unos datos específicos sea muy bajo. Lo que

habilita a la herramienta para realizar representaciones masivas de datos en cortos periodos de tiempos.

Scrutinizer [11] de Plixer es otro ejemplo de analizador de tráfico de red, al igual que la herramienta de Solarwinds, tiene una gran cantidad de funciones disponibles dependiendo de lo que esté dispuesto a pagar el usuario. Muchas de esas funciones coinciden. Scrutinizer permite al usuario exportar los datos a un archivo CSV, crear tableros en los que se crearán paneles para permitir distintas representaciones.

Todo el tráfico capturado de la red, en el caso de Scrutinizer se almacena en bases de datos de PostgreSQL, un sistema de administración de bases de datos relacional orientado a objetos. De esta forma, al igual que pasa con la herramienta de Solarwinds, se pueden realizar peticiones a las bases de datos y obtener una respuesta rápida, lo que permite su utilización con cantidades masivas de datos.

Existen también herramientas solo diseñadas para la representación de los datos almacenados en las bases de datos. Un ejemplo de estas herramientas gráficas es Kibana, un programa de la pila ELK [12] que expondremos en los siguientes apartados.

Este tipo de herramientas aportan al usuario grandes posibilidades para el análisis del tráfico de la red, otorgando la posibilidad de visualizar de una forma sencilla grandes cantidades de datos, que pueden representarse en gráficas de puntos, tablas, tartas y otras muchas posibilidades. Además de la representación, gracias a la conexión con colectores de flujo, estas herramientas permiten un filtrado relativamente rápido de todos los datos mostrados en el tiempo que el usuario ha seleccionado.

En los siguientes apartados se observará las distintas posibilidades de colectores e interfaces utilizadas en el desarrollo del programa y el equivalente más competitivo en la actualidad, la pila ELK. Además, dentro de los exportadores de flujo sin interfaz se encuentra YAF [13] *(Yet Another Flowmeter),* que se describe a continuación.

2.2 YAF y SILK

YAF es una herramienta, creada por el *Software Engineering Institute* de la *Carnegie Mellon University* de Estados Unidos, enfocada principalmente en la captura de grandes cantidades de tráfico en redes con grandes intercambios de información. YAF permite tanto la captura en tiempo real, como el procesamiento de datos almacenados. También tiene la capacidad de realizar un filtrado a la vez que realiza la captura, además de la posibilidad de captura de flujos unidireccionales.

Todas las acciones de YAF se manejan por medio del terminal, careciendo de una interfaz gráfica. Una de las características que lo hace especial es su formato de salida de capturas. YAF puede exportar los datos capturados a archivos con formato IPFIX o puede enlazarse con otro programa que reciba esos datos en formato IPFIX. Esto se realiza mediante el uso de una conexión en el puerto elegido por el usuario.

El archivo de configuración es editable por el usuario para poder adaptar el programa a sus necesidades, y así poder decidir el puerto de envío de los datos, para su posterior tratamiento. Usualmente esta herramienta se enlaza por el puerto 18001 con SILK [14], un programa desarrollado por el mismo grupo que se describe a continuación.

SILK *(System for Internet-Level Knowledge,* Sistema para el conocimiento a nivel de internet*)* es una colección de herramientas de análisis creadas por el CERT NetSA para facilitar el análisis seguro de redes o nodos con muy grandes cantidades de intercambios de información. SILK es una herramienta que permite una recolección eficiente de flujos, así como su almacenamiento y su posterior análisis mediante filtrado de los datos. Está enfocado principalmente, al igual que SILK, en el procesamiento de inmensas cantidades de datos.

Al igual que pasaba con YAF, todas las acciones de SILK se manejan mediante el uso de comandos por terminal. SILK necesita que se definan los puertos en los que va a escuchar, se pueden definir sensores, que son asignados a un puerto para su posterior enlace. Además de los puertos de escucha, SILK necesita también la ruta del directorio en el que serán almacenados todos los archivos de las capturas de flujos. Para el almacenamiento, SILK también utiliza el formato IPFIX, SILK separa en distintos directorios las capturas correctas y las capturas que, por culpa de los parámetros de entrada del usuario, han sido erróneas, lo que facilita al usuario el descubrimiento de errores.

Esta herramienta también tiene la capacidad de transformar archivos de un formato a otro, de tal forma que puede transformar, por ejemplo, una captura de tráfico de PCAP a un conjunto de registros de tipo IPFIX, e incluso puede seleccionar los campos deseados de un archivo CSV para formar un archivo IPFIX totalmente nuevo.

SILK tiene además la capacidad de realizar filtrados masivos de los datos almacenados. De manera que SILK filtrará todos los archivos que ha indexado previamente en distintos directorios dependiendo del día de almacenamiento y proporcionará como salida todos los paquetes que cumplan con las condiciones. Además de todas estas funciones, también tiene otras adicionales, tales como contar los paquetes que se han recibido en un tiempo determinado. Se aclarará con posterioridad el funcionamiento de las distintas capacidades de SILK cuando se vayan utilizando en el desarrollo de este TFG.

2.3 Pila ELK

La pila ELK es una agrupación o pila de distintas herramientas que permiten realizar de una forma segura la recolección de datos desde “cualquier fuente”, en “cualquier formato”, y realizar sobre estos datos, búsquedas y análisis, para visualizarlos en tiempo real en su interfaz gráfica. Esta pila está compuesta por herramientas que realizan distintas funciones para lograr el objetivo final. La pila ELK está formada por: Beats, Logstash, Elasticsearch y Kiban. El esquema general se puede apreciar en la Figura 2.3-1.

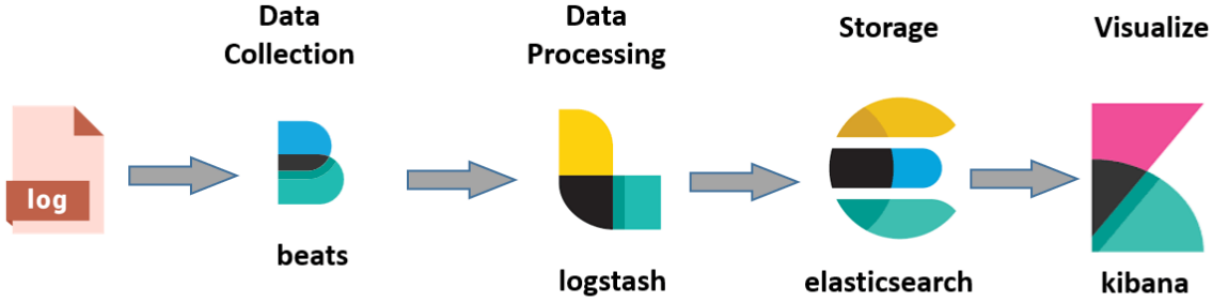

Figura 2.3-1 Diagrama del ELK stack [29].

2.3.1 Beats

Beats en un conjunto de herramientas que forman parte del *"ELK stack"*. Estas herramientas tienen una base común, son las encargadas de recoger datos desde una fuente para transmitirlos a otra herramienta del grupo *"ELK stack",* tales como Logstash o Elasticsearch.

Existen distintos tipos de Beats dependiendo del origen de los datos que se quiera extraer, siendo los más populares "Packetbeat" y "Filebeat". Packetbeat es una herramienta que permite la captura y exportación de una cantidad masiva de tráfico de red. La capacidad de extraer los datos desde "cualquier formato" como afirma la propia página web de la pila ELK, reside en la posibilidad de la creación de un Beat por el usuario.

2.3.2 Logstash

Logstash es una herramienta que tiene la capacidad de enlazar el proceso de captura y de indexación de los datos que se desean almacenar, mientras se realizan sobre ellos transformaciones y filtrados. Logstash actúa como un servidor, el usuario inicia Logstash a través del terminal, ya que carece de interfaz gráfica. En ese momento, Logstash comienza a esperar datos o comienza a leer datos de un archivo; acto seguido, procesa todos los datos recibidos según los parámetros establecidos por el usuario y, por último, los envía al destino que se haya seleccionado.

Todas estas elecciones que realiza Logstash son previamente establecidas por el usuario, creando un archivo de configuración ".conf". En este archivo el usuario debe establecer los parámetros básicos para el funcionamiento de Logstash:

- **Entradas.** En este apartado del archivo de configuración el usuario debe establecer las entradas que recibirá Logstash al ejecutarse, pudiendo ser un archivo almacenado anteriormente o un puerto para recibir los datos desde otra herramienta.

- **Filtros.** El apartado de filtros es el que hace esta herramienta realmente útil, el apartado de filtros nos permite establecer los campos que se leerán de un archivo, para realizar transformaciones de los tipos de datos a nuestra elección o para añadir nuevos campos a los datos procesados, tales como marcas de tiempo adicionales.

- **Salidas.** En este apartado Logstash nos permite definir el puerto de salida de la transmisión de datos. También nos otorga la opción de almacenar los datos en un archivo en el directorio elegido por el usuario. Si los datos van a ser indexados, nos permite definir qué campo será el que se utilice para indexarlos. Además, Logstash permite realizar asignaciones de campos a índices de última hora, e incluso permite la elección de otra dirección IP permitiéndolo enlazarse con un servidor externo.

2.3.3 Elasticsearch

Elasticsearch es el corazón de la pila ELK. Es el encargado de la indexación de todos los datos recibidos, siendo capaz de procesar una cantidad masiva de datos. Elasticsearch también es la herramienta encargada de realizar búsquedas de todos los datos almacenados según los filtros elegidos por el usuario. Hay que resaltar las diferencias entre este filtrado y el realizado por Logstash: mientras que Logstash filtra la recepción de los datos, decidiendo cuáles son los

que se transmiten a la salida o se desechan, Elasticsearch filtra todos los datos almacenados en la base de datos del usuario, pero no elimina los datos que no cumplen este filtrado.

Elasticsearch tiene la capacidad de recibir datos desde los Beats, Logstash o desde cualquier herramienta o servidor que transmita los datos en un formato compatible y transmita en el puerto definido por el usuario, que por defecto viene establecido en el 9200. El formato establecido es JSON. JSON es un formato de objetos inmensamente popular en la actualidad, ya que es el utilizado en el lenguaje de programación Javascript [15], utilizado mayormente en páginas web. El formato JSON indexa los datos de una forma sencilla de leer para el usuario, a la vez que fácil de procesar para computadoras, esto lo hace un formato sencillo y muy interesante para el intercambio de datos.

Elasticsearh también almacena los datos recibidos en el formato JSON, de este modo Elasticsearch procesa los datos extremadamente rápido, lo que permite al usuario realizar filtrados sobre datos masivos en un breve espacio de tiempo. Una cualidad de Elasticsearch es que tiene la posibilidad de crear varios índices para que el usuario pueda almacenar distintos datos separados por índices únicos y pudiendo procesar sólo los datos seleccionados.

Elasticsearch se ejecuta desde el terminal, aunque puede observarse un breve resumen si accedemos en un navegador a la IP y al puerto que hayamos establecido, pudiendo incluso mediante el uso de algunos "*Plug-Ins*" representar gráficas de los datos. Pero estas representaciones son muy limitadas, por eso la mayoría de los usuarios utilizan Kibana para la representación.

2.3.4 Kibana

Kibana es una herramienta que permite visualizar los datos y navegar por los distintos índices que hayan sido creados. Por definición, Kibana es la interfaz gráfica ideal para Elasticsearch, pudiendo crear distintos paneles para la representación, como por ejemplo histogramas, gráficos de líneas, tartas y muchas más posibilidades.

Manejar la interfaz gráfica de Kibana es tan sencillo cómo dirigirse a la URL de la dirección IP [16] donde esté instalada y acceder al puerto 5601. Una vez allí, se tiene acceso a sus diversas funciones, desde crear un nuevo panel para visualizar datos hasta acceder al resumen de los valores de los índices, donde se proporciona información general de los datos guardados en Elasticsearch.

En la Figura 2.3-2 se puede observar un ejemplo de la visualización de distintos datos de un índice previamente guardado. Cada panel es independiente y se pueden añadir tantos paneles como el usuario prefiera. Además, se puede apreciar a la izquierda las distintas opciones de las que dispone Kibana para nuestros datos, pudiendo visualizar, indexar a través de Elasticsearch e incluso utilizar comandos para bases de datos (PUT, DELETE, etc.) en su consola para manejar todos los índices.

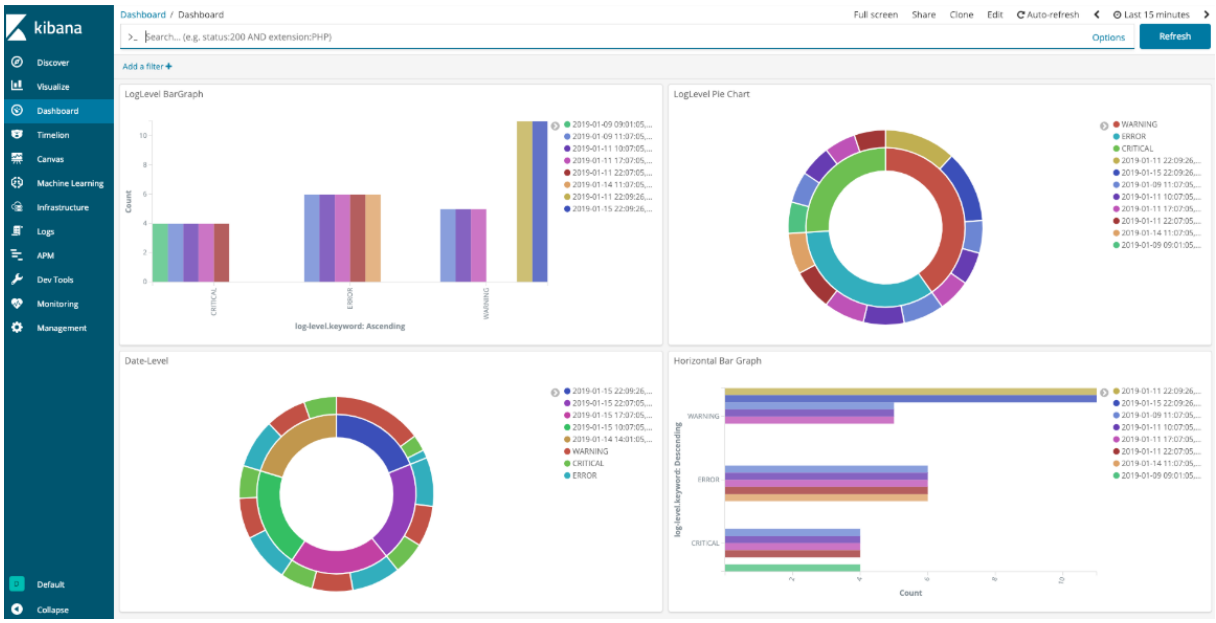

Figura 2.3-2 Ejemplo de un tablero de Kibana [28].

Hay que destacar que Kibana realiza todas las operaciones mediante el uso de Elasticseach, no pudiendo estas realizarse por sí solas. Kibana proporciona una forma sencilla para observar los datos al usuario, pudiéndose realizar incluso gráficos a tiempo real que se recargan cada intervalo de tiempo establecido. A diferencia de las herramientas anteriores como Beats o Logstash, Kibana no requiere de acciones por el usuario en los archivos de configuración, a no ser que, el usuario no pretenda utilizar los valores que están establecidos por defecto.

Por último, cabe destacar que Kibana está diseñada especialmente para enlazarse con la pila ELK, y optimizar al máximo todas las acciones que el usuario puede realizar sobre los datos, proporcionándole muchas funcionalidades y un control casi completo sobre todo los procesos que ocurren en la pila ELK. En cambio, existen otras herramientas para visualizar cantidades masivas de datos, que permiten enlazar como fuentes a una vasta cantidad de colectores. La herramienta más popular en la actualizad es Grafana.

2.4 Grafana

Grafana [17] es la herramienta de visualización de datos más extendida actualmente. Grafana tiene la capacidad de recibir datos desde una inmensidad de fuentes, pudiendo enlazarse con la mayoría de los colectores de flujo actuales. Este enlace se realiza a través de su interfaz, a la que sea accede en el navegador con la IP y puerto seleccionadas, que por defecto es el puerto 3000. El usuario tiene la capacidad de crea varias fuentes de datos para enlazar sus distintos programas. Esta acción es relativamente sencilla, ya que Grafana guía al usuario en la creación de su primer panel.

Para soportar las distintas fuentes, Grafana necesita de la instalación de ellas, acción que se realiza mediante su página WEB oficial. Una vez creadas las fuentes para nuestras representaciones, Grafana permite crear varios paneles de visualización, pudiendo instalar paneles adicionales en la misma página que las fuentes. Una característica que destaca de Grafana es que permite la representación de datos de varias fuentes en una misma interfaz; esta

propiedad la convierte en la herramienta ideal para el caso de ser necesarios varios programas para realizar una tarea.

Al igual que en la herramienta de representación Kibana, Grafana también permite la visualización a tiempo real, actualizando los paneles cada intervalo de tiempo definido por el usuario. Cada uno de estos paneles creados por el usuario es editable casi completamente, desde las variables que se representan hasta la apariencia del propio panel.

Cada vez que se crea o se actualiza un panel, Grafana manda una petición a la fuente de datos a la que esté conectado, pasándole como parámetros todas las variables que el usuario haya definido, que tendrán que ser de al menos una, el tiempo. Después de solicitar los datos, Grafana procede a la escucha en el puerto establecido al declarar la fuente de datos. La transmisión entre Grafana y la fuente se realiza mediante objetos JSON, tal y cómo hemos visto con Elasticsearch.

Todas las variables que Grafana proporciona a la fuente de datos las define el usuario, pudiendo crear varios tipos de variables, dependiendo del contenido que vayan a tener, esto hace que la interfaz sea compacta y de gran utilidad.

En la Figura 2.4-1 se puede observar un ejemplo de la interfaz de Grafana, donde se puede observar cada uno de los paneles que se permiten crear al usuario, así como, mover y dimensionar a su gusto. Arriba a la derecha está situada la variable del tiempo, que la fuente de datos utilizará para responder con los datos adecuados.

Figura 2.4-1 Ejemplo de un tablero de Grafana [30].

En estos paneles, Grafana también permite la colocación de alertas, es decir, se puede configurar una alerta de Grafana para que cuando cierta gráfica, tabla o tarta llegue a un valor determinado, Grafana genere una alarma y nos avise de qué la está produciendo. Esto puede ser de gran utilidad en el caso de tener una gran cantidad de paneles y no tener la posibilidad de vigilarlos continuamente.

En el caso de que el usuario haya enlazado mal la fuente de datos, o que el envío de los datos esté dando algún error, Grafana no dibujará nada, pero en la esquina superior izquierda aparecerá un símbolo de alerta que nos indicará que algo está fallando. Si se coloca el cursor sobre esa alerta visual, Grafana proporciona al usuario toda la información de por qué puede estar produciéndose ese error, que suelen ser debidos a datos fueras de rango de tiempo, a un formato inadecuado de los datos o, debido a que no ha recibido dato alguno. Un ejemplo de error es leer de una fuente de datos incorrecta como se muestra en la Figura 2.4-2.

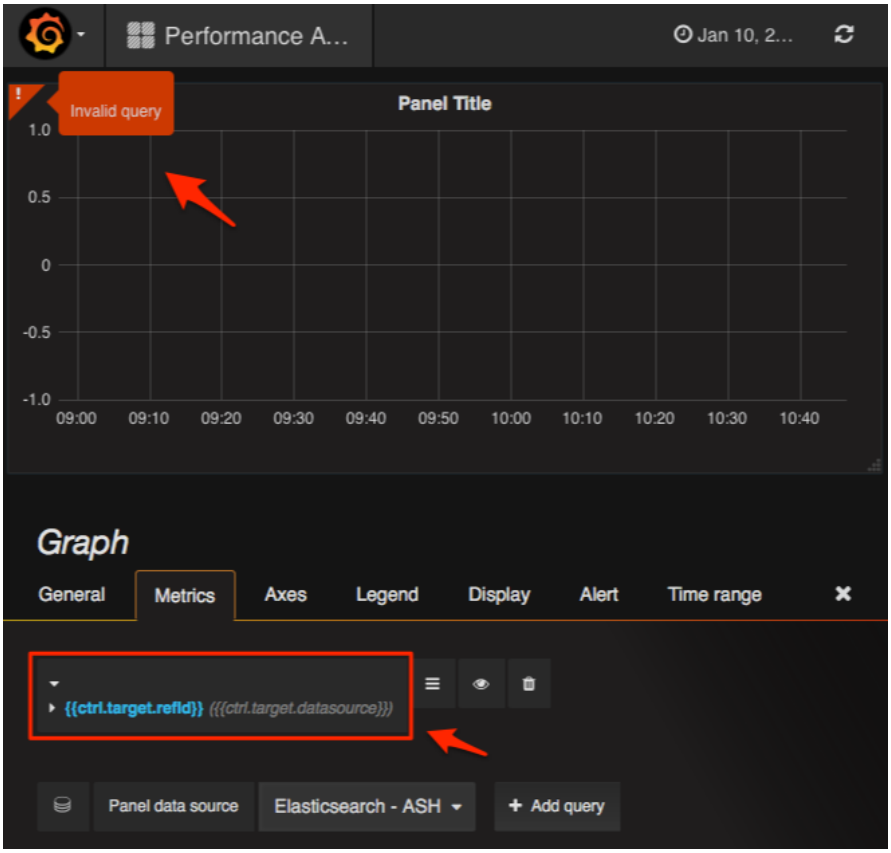

Figura 2.4-2 Ejemplo de error en un panel de Grafana

2.5 Conclusiones

El gran aumento del tráfico de datos en los últimos años ha causado que los métodos tradicionales de análisis de tráfico de la red, centrados en la obtención de mucha información de cada paquete, hayan pasado a estar obsoletos, siendo reemplazados por métodos y herramientas de procesamiento masivo de datos. También se ha producido un cambio del formato respecto a cómo se comparte la información entre distintos programas, derivando a un JSON, debido a su rapidez para su procesamiento y visualización.

En este capítulo hemos repasado el estado del arte sobre el que realizaremos el trabajo. Viendo una definición general de los colectores de flujo y una visión particular de las herramientas que serán utilizadas a lo largo de esta memoria. En particular, nos centraremos sobre los programas YAF y SILK, ya que además de centrarse en la velocidad de las capturas, filtrados y transmisiones, también ofrecen posibilidades para reducir el tamaño del almacenamiento. Reducir el tamaño del almacenamiento para poder albergar una cantidad de datos masiva será el principal objetivo de este Trabajo Fin de Grado.

3 Diseño

El objetivo principal de este capítulo será el de proporcionar una visión clara de las decisiones que se han tomado en la selección de las herramientas a utilizar. Se procederá a analizar las distintas posibilidades existentes para la realización del trabajo, observando las diferencias entre las herramientas, y las posibilidades que ofrecen.

Principalmente, estará enfocado hacia los distintos formatos para las transmisiones, la capacidad de enlazarse con herramientas, y las posibilidades de filtrado o almacenamiento que otorgan al usuario.

3.1 Formato de almacenamiento

En la actualidad, existe una amplia cantidad de formatos de almacenamiento para el tráfico de la red. Es suficiente realizar una captura en Wireshark y proceder a almacenarla para observarlo. Las diversas opciones en el formato de almacenamiento aportan una gran facilidad para utilizar los datos almacenados en un sinfín de herramientas El caso más claro es el de PCAP, casi admitido por cualquier herramienta de almacenamiento o procesamiento de tráfico de red.

Sin embargo, este tipo de formato no podrá ser utilizado para la realización de este trabajo, debido al gran tamaño que ocupa el almacenamiento de los datos en este tipo de formatos, ya que almacena el contenido y este no será necesario en el sistema. Esto conduce a evaluar como opción preferente, el uso del formato JSON para la realización del trabajo. JSON es un formato que, tal y como se ha expuesto en capítulos anteriores de esta memoria, facilita la lectura de los archivos para el usuario, a la vez que es rápido de procesar por las herramientas de tráfico de datos, y, además, el formato en la transmisión de estos programas suele ser realizado mediante el mismo formato, por lo que no se debería de transformar el tipo de datos al ser enviado.

JSON podría parecer el candidato ideal para la realización del trabajo, facilitando así la conexión entre las distintas herramientas y la velocidad de procesamiento de estas, pero, como se ha comentado, el objetivo principal de este trabajo es el funcionamiento correcto del programa de visualización priorizando totalmente la reducción del tamaño de los datos almacenados, debido a que se aspira a un almacenamiento masivo de ellos, el formato del archivo debe perseguir ocupar el menor espacio posible.

Este razonamiento no deja casi sin ninguna opción, pero también recuerda la otra opción más básica, IPFIX (*IP Flow Information Export*). IPFIX es un protocolo de transmisión de registros de flujos de red que tiene un funcionamiento de empuje: el emisor envía cada cierto tiempo predefinido un mensaje para configurar a los receptores, que no envían ningún mensaje de respuesta. La velocidad del protocolo de transmisión de IPFIX sería comparable a la velocidad de procesamiento del formato JSON, pero la principal diferencia entre los dos está en el tamaño de almacenamiento. Esto es debido a la diferencia de disposición de los datos en el archivo: mientras que JSON indexa variables dentro de apartados, que a su vez están dentro de otros apartados definidos por llaves, estando cada variable igualada a un valor entrecomillado, IPFIX indexa los datos de una forma totalmente distinta, de manera que sólo define una vez los campos al enviarle la configuración al receptor, y después todos los datos van en formato binario.

Esta sutil diferencia hace que, en la realización de este trabajo, se utilice preferentemente el formato IPFIX en el almacenamiento, y se analizarán las diferencias entre los distintos formatos para comprobar si realmente aporta mejoría.

Por supuesto, en la búsqueda de que ocupe el mínimo espacio, hay que tener además en cuenta el formato CSV, que también utiliza texto plano para el almacenamiento, aunque según observaciones realizadas, IPFIX debería de ocupar siempre menos que el formato CSV, ya que CSV incluye un delimitador entre los campos, suele estar delimitado mediante comas y también incluye una cabecera con los campos de los datos, además, los campos no son binarios como en IPFIX, sino, texto plano.

3.2 Elección del colector

Al decidir el formato en que se va a almacenar las capturas de tráfico, también se disminuyen las opciones en la elección del colector de flujo que se va a utilizar. Con esta restricción, es necesario encontrar un colector que permita el almacenamiento en formato IPFIX, pero también cabe la posibilidad del enlace entre un colector y una herramienta que nos permita almacenar y trabajar con cantidades masivas de datos en el formato deseado.

Ante esta decisiva elección, lo primero es analizar en los procesos y funciones que suelen realizar otros sistemas similares al que se pretende diseñar. El principal sistema de este tipo en la actualidad es la pila ELK. En el sistema que se pretende diseñar, hay que ejecutar las acciones que realizan tres de ellos, los Beats, Logstash y Elasticsearch. Aunque, se puede obviar la herramienta Logstash, debido a que no se pretende filtrar en la recepción, sino sobre los datos previamente almacenados.

Al tener claras las restricciones y necesidades de nuestro sistema, se procede a analizar el comportamiento de los Beats de la pila ELK. Los Beats, son herramientas que permiten la captura de datos desde distintos tipos de fuentes, desde archivos o desde la propia red. Al pretenderse diseñar un sistema que utilizará el flujo real de la red, se analiza en detalle el Beat encargado de esta función, Packetbeat. La herramienta Packetbeat está especialmente diseñada para monitorizar servicios y aplicaciones a tiempo real.

Packetbeat realiza una serie de pasos para recibir, procesar y enviar los datos a la herramienta a la que este enlazada, además captura el tráfico de la red o el servicio que tengamos configurado, decodifica las transmisiones para poder procesarlas, extrae los campos de todos los datos recibidos y por último genera un archivo JSON indexándolo para transmitir los datos a la herramienta a la que esté enlazado.

La elección del diseño será la de una herramienta que siga todos estos pasos, que se diferencie en el formato de envío y que sea IPFIX, debido que así se perderá el menor tiempo posible al transformar los datos a este formato, y se podrá capturar todo el tráfico de la red, por muy masivo que este sea. La elección, por tanto, podría ser la de utilizar YAF en la recepción de los datos.

La razón de elegir YAF se debe a que, al igual que Packetbeat, proporciona la capacidad de procesar una cantidad masiva de datos a una gran velocidad. YAF realiza prácticamente las mismas acciones, y además captura todo el tráfico que circula por una interfaz deseada por el usuario, pudiendo elegir entre los distintos interfaces que el usuario tenga disponible. Al capturar el tráfico, decodifica los paquetes para poder procesarlos, y una vez que los ha

decodificado, continúa extrayendo los campos que se desean analizar, y que además pueden ser decididos por el usuario. Por último, YAF envía todos los registros por cada flujo de red que hayan sido observados y procesados a través del puerto seleccionado por el usuario, en el formato IPFIX. Esta diferencia, permite tomar una decisión para el diseño del sistema a desarrollar, YAF es el mejor candidato para la recepción de los datos del sistema.

De este modo, YAF se podrá utilizar para capturar y enviar los datos del tráfico de la red a la herramienta que después almacenará y podrá filtrar una vez que todos los datos estén almacenados, y por supuesto lo realizará en el formato IPFIX. Para decidir la herramienta de este paso, hay que analizar los procesos que realiza Elasticsearch para almacenar y filtrar los datos que tiene almacenados. Elasticsearch recibe todos los datos en formato JSON a través del puerto en el que esté configurado, por defecto el puerto 9200.Con todos los datos que recibe, Elasticsearch crea un nuevo índice, que, para el caso de tráfico de red, estará indexado por el tiempo en el que se ha recibido el paquete. Una vez tiene creado el índice, comienza a indexar todos los datos que recibe, acumulando uno tras otro dentro del mismo índice, almacenando todos los datos en formato JSON, por supuesto.

Teniendo todos los datos almacenados en archivos JSON, Elasticsearch permite además realizar búsquedas en ellos de valores concretos, filtrando así la cantidad masiva de datos que puede tener almacenados y entregando simplemente los datos deseados. Esto lo realiza en un tiempo muy reducido, incluso cuando se tiene una cantidad ingente de datos. De nuevo, el principal problema de la utilización de Elasticsearch para nuestro sistema es el formato en el que almacena las capturas de tráfico, no pudiendo ser en JSON. Buscando una herramienta con las mismas funcionalidades de Elasticsearch, pero con esta sutil diferencia, se encuentra SILK.

SILK proporciona al igual que Elasticsearch la capacidad de recibir datos desde otra herramienta, sólo que, con diferencias, la más obvia es el puerto por defecto, que en SILK es el 18001 y por supuesto el formato en el que se recibe la transmisión de los datos desde la herramienta que captura el tráfico de la red, siendo el formato IPFIX. De este modo, se obtiene lo que se pretendía, un enlace de herramientas en el mismo formato, para evitar pérdidas de tiempo al transformar los datos de un formato a otro.

Una vez que SILK recibe los datos desde la herramienta a la que está enlazado, SILK los va almacenando en un archivo hasta que se cumple el intervalo de tiempo definido por el usuario, creando un nuevo archivo de guardado. Con todos los datos que tiene almacenado, SILK permite realizar filtrado sobre ellos al igual que Elasticsearch, con la diferencia de que SILK no distingue por índices porque en IPFIX no pueden existir, así que, al realizar un filtrado para encontrar los datos deseados, SILK tendrá en cuenta todos los datos almacenados. Dado que cada cierto tiempo se genera un nuevo archivo, este filtrado a SILK no le lleva prácticamente tiempo, pudiendo competir en velocidad con el filtrado de Elasticsearch.

Además, SILK tiene una variedad de funciones que pueden ser de gran utilidad, desde la construcción de archivos IPFIX desde PCAP o CSV, hasta funciones propias que utilizan BASH [18] para realizar ciertas acciones, como contar la cantidad de paquetes que han sido recibidos en un intervalo de tiempo. Esta ventaja permitirá que, al utilizar este tipo de funciones, el tiempo de ejecución disminuya en gran parte.

Para la configuración de la fuente de los datos en SILK, previamente hay que configurar un archivo de sensores llamado “sensors.conf”, en el que se pueden establecer los puertos en los que SILK escuchará para recibir todos los datos, este proceso es rápido e intuitivo y probar la

configuración para evaluar su funcionamiento correcto, conociendo que si existe algún error SILK almacenará los datos en un directorio de errores y no en el directorio donde se almacena normalmente los datos si no ha ocurrido ningún percance.

En este punto del diseño se va vislumbrando de una forma más intuitiva el sistema buscado, dado que se pretende utilizar un enlace entre SILK y YAF para la recolección de los datos de la red y SILK para el filtrado de todos los datos almacenados, teniendo así un enlace teóricamente correcto.

3.3 Servidor de enlace

Debido a la decisión de realizar el enlace y almacenado en formato IPFIX, se desemboca en la utilización de un enlace de YAF y SILK como colector de flujo.Esto lleva a un problema crucial, SILK no está soportado por prácticamente ninguna herramienta de representación masiva de datos.Para poder enlazar SILK y una herramienta para la posterior visualización, no se puede utilizar ninguna característica de SILK porque básicamente, no funcionarían ya que SILK no puede recibir peticiones e incluso si lo hiciese devolvería los datos en el formato IPFIX, lo que no suele estar soportado.

Para poder enlazar SILK con la herramienta elegida se necesita implementar un servidor HTTP [19] que actúe de puente entre ellos, pudiendo así comunicarse de una forma óptima. Sabiendo que se necesita de un servidor para la comunicación, la primera cuestión que aparece es el lenguaje de programación a utilizar en dicho servidor. El lenguaje de programación elegido debería ser uno que permitiese realizar todas las operaciones necesarias para el enlace y la comunicación entre las dos partes. El lenguaje C [20] es sin duda una opción que siempre hay que tener en cuenta debido a su poco consumo de memoria y su velocidad para realizar todas las operaciones, pero C no es un lenguaje que facilite la implementación de servidores. En C para implementar un servidor se necesita desarrollar cada uno de los *sockets* a mano, además de la administración de todas las acciones y de todos los protocolos subyacentes en una comunicación HTTP, desde codificar los mensajes en envío y decodificarlos en recepción, hasta incluso voltear el contenido binario los paquetes porque vienen girados desde la red.

Una vez descartado el lenguaje de programación C, debido a su alta complejidad en las acciones y a que probablemente se deberían realizar numerosas funciones complejas a lo largo del programa, se pasa a la siguiente opción. En la búsqueda de un lenguaje que nos proporcione facilidad para crear un servidor HTTP y que nos permita utilizar funciones de muy alto nivel, se llega al lenguaje de moda, Python. Python es el lenguaje, con Javascript, que está creciendo más rápido en la cantidad de usuarios. Esto se debe a la posibilidad de crear funciones y compartirlas con el resto de los usuarios, además de la opción contraria, descargar funciones de alto nivel que otros usuarios hayan diseñado anteriormente y que sean de gran utilidad. La facilidad para descubrir nuevas y útiles funciones acompañado de su limpia apariencia lo hacen un candidato muy bueno para la realización del trabajo. Python aportará todas las funciones necesarias para la conexión de nuestro sistema de recepción de datos, y la herramienta con la que se pretende representar gráficamente.

También hay que recordar que las dos herramientas seleccionadas para la recolección de los datos carecen de interfaz gráfica y deben ser controladas por terminal, así que todas las acciones que realizan YAF y SILK deberán ser realizadas mediante el lenguaje de programación BASH. De hecho, todas las acciones que lo permitan se harán en BASH, debido a que mejorará en gran medida el tiempo, ya que Python utiliza una gran cantidad de memoria y además es

relativamente lento al realizar las operaciones, al ser un lenguaje interpretado y de muy alto nivel.

3.4 Plataforma de visualización

En este punto, ya se tienen seleccionadas las herramientas para realizar la captura del tráfico de la red, y el lenguaje de programación sobre el que se va a trabajar, para la habilitación entre las comunicaciones desde SILK a la plataforma de visualización. Teniendo esto establecido, se procede a continuación a analizar las posibles plataformas de visualización existentes, para elegir la plataforma que se adapte a las necesidades de forma óptima.

Para una correcta elección en este paso también se debe prestar especial atención a las distintas opciones disponibles. El sistema necesitará de una representación de los datos almacenados como la que hace Kibana, que podría ser una opción el sistema.

Kibana es la herramienta de visualización de la pila ELK, recibe los datos desde la herramienta Elastcisearch y los utiliza para realizar sus gráficas y representaciones. Todos los datos los recibe en formato JSON. Kibana aporta gran facilidad al representar los datos, creando cada panel de forma intuitiva y guiada por el mismo Kibana. El usuario se dirige a la pestaña de creación de un nuevo tablero, una vez allí, Kibana requiere al usuario el índice que desea representar e incluso le propone que si carece de índices puede utilizar los que vienen por defecto de prueba. Una vez seleccionado el índice, Kibana examina el JSON que Elasticsearch tiene almacenado y extrae todos los campos de los datos que están contenidos. Una vez los ha extraído, los enseña al usuario para darle la opción de utilizar alguno de ellos como índice para posteriormente utilizarlo para las representaciones.

Habiendo seleccionado el índice deseado, en el tablero se puede representar una amplia variedad de paneles o gráficos. El índice seleccionado se utilizará para variar la posición del eje de las abscisas. De este modo, el usuario puede observar los datos de los índices que desee mientras varía el índice utilizado para la indexación. Es común que este índice sea un valor temporal, de esta forma las gráficas quedarán representadas frente al tiempo y será más fácil de observar para el usuario. Si se carece de valores temporales, este índice puede ser sustituido por cualquier otro.

Todos los paneles que se puedan representar se podrán editar casi en su totalidad por el usuario, permitiendo editar los campos representados, agregar más de un campo a representar o reduciendo o aumentando el tiempo de representación. Kibana proporciona una amplia libertad a la hora de representar los datos deseados y otorga una gran variedad de herramientas, al igual que funciones matemáticas para obtener valores medios, máximos y mínimos automáticamente. Sin embargo, esta herramienta tiene una difícil conectividad con herramientas que no sean Elasticsearch, debido a que está especialmente diseñada para él, y no siendo capaz de enlazarse con ninguna otra. Por ello, el servidor HTTP no podrá enlazarse con Kibana.

Ante esta realidad, se procede a buscar una plataforma de representación que tenga capacidad de conectarse con un servidor HTTP. Claramente, la opción a tener en cuenta es Grafana, ya que además es el programa de representación más popular en la actualidad. Grafana tiene algo que Kibana no tiene, la posibilidad de instalar nuevos “Plug-Ins” para enlazarse con otras fuentes de datos. Partiendo de esta base, se puede comparar las posibilidades y capacidad de Grafana frente a las de Kibana. Al igual que Kibana, Grafana aporta un total control de los datos

representados, pudiendo seleccionar la fuente de los datos que se pretenden representar, incluso permitiendo representar datos desde varias fuentes distintas en un mismo cuadro de mando. Grafana, frente a Kibana, no necesita que el usuario defina un índice, sino que al realizar las peticiones a la fuente datos simplemente los recibe y representa. Los paneles de Grafana también permiten al usuario el total control sobre los campos a representar, e incluso permite variar el tamaño y posición de estos libremente. Además, Grafana posee una amplia comunidad que aporta nuevos *"Plug-Ins"* que permiten realizar más y más acciones desde la misma plataforma. Se podría decir que Grafana crece cada día más y más, debido a la interacción de sus usuarios, que incrementan las posibilidades de la plataforma de una forma colosal.

La interfaz de Grafana es más intuitiva para el usuario y facilita la utilización óptima de la herramienta. En pocas horas se puede llegar a saber utilizar la mayoría de las funciones que posee, y además cada uno de los *"Plug-Ins"*, viene con una documentación adjunta para facilitar la comprensión y la instalación.

De esta forma, el sistema se encuentra en el punto de diseño en el que ya posee un generador y colector de flujos formado por YAF y SILK, que interactúan mediante el protocolo IPFIX, los datos pasan entonces a un servidor HTTP construido sobre el lenguaje de programación Python, y posteriormente se pretende enlazar ese servidor con la plataforma de representación Grafana.

3.5 Enlace con la plataforma de visualización

El sistema está diseñado casi al completo, pero necesita un enlace entre el servidor HTTP y Grafana. La primera opción es buscar entre todas las fuentes de datos que soporta, la opción para seleccionar como fuente a la herramienta SILK, esta opción ahorraría la creación del servidor HTTP en su totalidad, pero al ser SILK una herramienta tan poco utilizada, Grafana no posee de la opción de la utilización de SILK como una fuente de datos.

En la página WEB de Grafana se puede encontrar una amplia variedad de "Plug-Ins" que se pueden instalar para utilizar funciones que no soporta Grafana por defecto. La instalación de estas ampliaciones de Grafana se realiza mediante el terminal, es decir, con el lenguaje BASH. Una vez instalada tan sólo hay que reiniciar el servicio de Grafana para que funcione correctamente. La búsqueda de un "Plug-In" deseado es muy sencilla gracias a la división por tipos que posee Grafana y a la posibilidad de realizar una búsqueda de texto que permite acotar todavía más los resultados.

Al no existir esta solución, la siguiente opción lógica es la utilización de una fuente de datos que soporte la trasmisión de datos mediante el protocolo de transmisión del tráfico de red IPFIX. Este enlace permitiría que el procesamiento que deba realizar el servidor HTTP sea casi nulo, y, por tanto, la velocidad de las representaciones sea muy elevada. Por desgracia, Grafana no posee ninguna opción para la utilización del protocolo IPFIX en el enlace, como era de esperar.

De este modo, se pasa a analizar la fuente de datos de Elasticsearch, ya se ha comentado que hay bastantes diferencias con el sistema que se pretende diseñar, aunque las bases son prácticamente las mismas. Esta fuente de datos viene instalada en versión por defecto de Grafana, ni siquiera es necesario implementarla. Analizando los campos requeridos para la conexión con nuestro servidor se encuentran campos como la IP y el puerto, que, si pueden ser utilizadas por nuestro servidor, pero si se continúa analizando se descubre el mayor problema. El enlace con Grafana necesita del nombre del conjunto indexado y además necesita del índice

de tiempo. El servidor HTTP no puede proporcionar esa información debido a que en el sistema a desarrollar se busca que ese exceso de espacio ocupado no exista. Se pasa a la búsqueda de una fuente de datos estándar, que no pertenezca a ninguna herramienta en concreto.

En esta búsqueda se encuentra que la única fuente de datos que no pertenece a un programa en concreto es la fuente de datos JSON [21] (no confundirla con los objetos JSON). Esta fuente de datos hace que Grafana envíe peticiones en el formato JSON, de ahí su nombre. Una vez instalada la fuente, se procede a establecer los parámetros de la IP y el puerto al que Grafana se enlazará. Al ser una fuente de datos estándar, Grafana no requiere de más parámetros, esto permite que el servidor pueda enlazarse de forma correcta.

Al elegir esta fuente de datos, debido a la ausencia de otras, el servidor HTTP recibirá las peticiones de Grafana en el formato JSON, pero también será necesario que responda a estas en el mismo formato. Gracias a la elección del lenguaje de programación Python, esto será posible llevarlo a cabo sin mayores problemas.

Como curiosidad, hay que destacar que durante la realización del trabajo se produjo la publicación de una nueva fuente de datos que se comunicaba mediante objetos JSON. La fuente de datos utilizada en el sistema final diseñado es la versión **actualizada (JSON)**, puesto que la fuente de datos inicial era SimpleJSON [22].

3.6 Conclusiones

En este momento, se ha conseguido diseñar y construir el sistema completo ilustrado en la Figura 3.6-1. El sistema estará formado por el enlace de YAF y SiLK en la recepción, que almacenarán todos los datos extraídos de la captura de la red, comunicándose mediante el protocolo IPFIX. Este colector se conectará a un servidor HTTP construido sobre el lenguaje de programación Python, después el servidor se comunicará con Grafana para habilitar las peticiones a nuestra base de datos identificada con la etiqueta “PC” en el diagrama. Toda la comunicación entre el servidor y Grafana se realizarán con objetos JSON mediante el aprovechamiento de su fuente de datos JSON. De esta manera se logra alcanzar el diseño del sistema deseado.

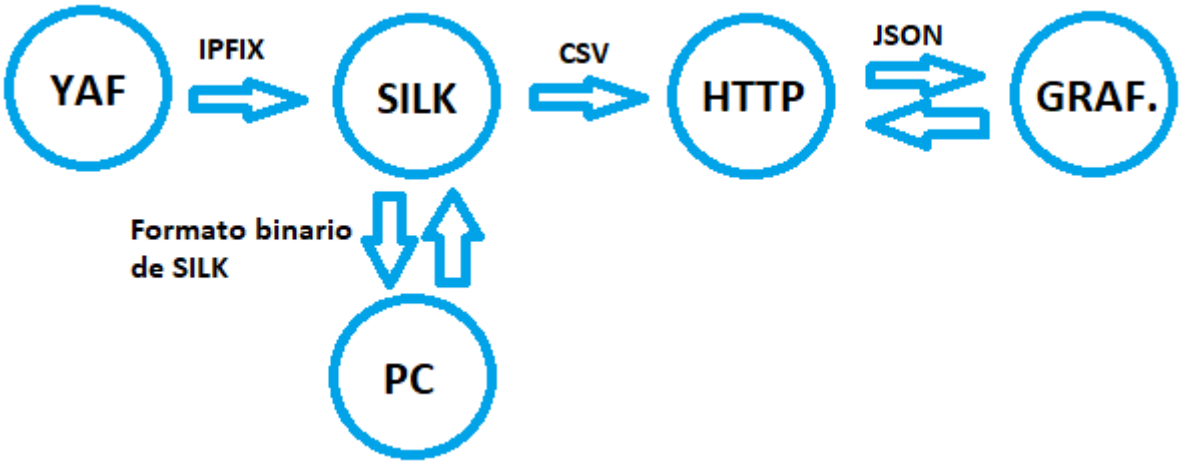

Figura 3.6-1 Diagrama del sistema desarrollado

Se logra así un diseño que cumple con todos los requisitos necesarios para la mejora de la pila ELK en una reducción del espacio necesario para el almacenamiento, y debido al procesado en el protocolo IPFIX, se puede alcanzar una velocidad de la transmisión de datos que sea comparable. Todo este sistema en teoría cumple esos requisitos, pero en los siguientes capítulos se expondrán las implementaciones prácticas sobre este sistema diseñado.

4 Desarrollo

En este capítulo se presentan las herramientas desarrolladas a lo largo de la realización del trabajo. En la exposición del desarrollo de las herramientas se presta especial atención a los detalles que son de gran importancia para el funcionamiento del sistema. Dado que en la realización de los enlaces entre las distintas herramientas utilizadas se han llevado a cabo cambios en los archivos de configuración, estos cambios también se han reflejado en este capítulo.

Para facilitar una visión generalista del sistema completo desarrollado, se seguirán los mismos pasos que se han descrito en el apartado de diseño, comenzando desde la captura de los datos hasta la representación final.

4.1 Captura de flujos

En esta parte del sistema, se pretende obtener un enlace estable y fiable entre las herramientas YAF y SiLK. El enlace se creará a través del puerto 18001, transmitiendo los datos en el formato IPFIX. Para habilitar este enlace entre las dos herramientas ha sido necesaria la creación de un archivo de configuración de sensores llamado “sensors.conf” que le indica a la herramienta rwflow (herramienta utilizada por SiLK para el enlace) el puerto y la dirección IP desde donde tiene que realizar las capturas de los datos.

Además de la creación del archivo de configuración para el enlace, también es necesaria la creación del archivo “SiLK.conf”, en el que se establecen los sensores que serán habilitados entre todos los que están creados, y la lógica para empaquetar los datos recibidos a través de los sensores configurados.

Una vez creados todos los archivos de configuración necesarios para la captura del tráfico de la red, ya pueden utilizarse para realizar la funcionalidad del almacenamiento. Todos los comandos utilizados en el colector de flujo serán expuestos en el siguiente capítulo, puesto que, en ese caso, no se han desarrollado, sino que se han utilizado.

4.2 Creación del servidor

En este apartado se va a presentar el desarrollo del servidor HTTP implementado con el lenguaje de programación Python. Debido a su extensión, no se aportarán capturas de fragmentos del código en este apartado. Para poder comprenderse de una forma lógica, el desarrollo del servidor será expuesto en el orden de ejecución del código, es decir desde que Grafana realiza una petición para uno de los paneles, hasta que recibe los datos extraídos desde el almacenamiento generado por el colector de flujo formado por YAF y SiLK.

4.2.1 Conexión TCP

Para enlazar correctamente el sistema de almacenamiento de los datos y el sistema de representación de los mismos, se requiere de una conexión TCP [23] entre ellos, que habilite la transmisión de las peticiones por parte de Grafana, y que estas peticiones a su vez sean transmitidas directamente a SiLK, para que pueda proporcionar los datos deseados por el usuario.

Este enlace se lleva a cabo mediante Python, utilizando el módulo “http.server”, con el que se obtienen las funciones necesarias para el establecimiento de un servidor HTTP. Para ello se necesitará la creación de las variables que contendrán la dirección IP, y el puerto en el que se pretende realizar el enlace. En el caso del sistema desarrollado, se utilizará el puerto 9000, al ser un puerto que ninguna de las herramientas del sistema utiliza.

Una vez que se han creado las variables, el servidor HTTP esperará peticiones desde el puerto 9000 de forma indefinida. Al recibir una petición, el servidor analiza si se trata de una petición GET o de una petición POST, que son las dos peticiones que puede emitir Grafana para la fuente de datos JSON. La principal diferencia entre estas dos peticiones, a parte del tipo de petición, es que Grafana las utiliza para distintas funciones. Una petición GET es un tipo de petición del protocolo HTTP que se envía para que el servidor responda con un archivo o cualquier otra información requerida. En el caso del sistema desarrollado, Grafana envía una petición GET al establecer una nueva fuente de datos, para comprobar si esa fuente de datos se encuentra activa, o en cambio, no se puede enlazar con ella.

El servidor al recibir la petición GET desde Grafana le responderá con el mensaje 200 OK, enviándolo en formato texto de HTML [24]. Esta respuesta del servidor en el protocolo HTTP significa que todo está correcto, es decir, el enlace se ha realizado de forma satisfactoria.

4.2.2 Procesamiento de peticiones

Todas las peticiones de datos que Grafana realizará al servidor tienen algo en común: todas ellas utilizarán el tipo POST. Una petición POST es un tipo de petición del protocolo HTTP, que se envía al servidor para que este escriba o almacene cierta información. En el caso de la fuente de datos JSON, estas peticiones se utilizan para requerir al servidor que le devuelva los datos con los que realizará las representaciones. Cada panel de Grafana tiene diferencias con los demás, debido a esto, es necesario conocer los paneles de Grafana que el servidor es capaz de enlazar forma correcta con la base de datos antes de pasar a ver las diferencias de procesamiento entre cada uno de ellos.

A continuación, se muestran todos los paneles soportados por el servidor HTTP desarrollado, acompañado de una breve descripción de los mismos.

- **Gráfico temporal:** Este tipo de paneles está soportado en la versión de Grafana por defecto, no es necesario su instalación. Sirve para representar series temporales, pudiendo establecer el eje de abscisas donde el usuario desee. Este tipo de representación será de gran utilidad para poder observar la velocidad en la que los datos fueron recibidos, pudiendo representar en el caso del sistema varios tipos distintos de este panel, dependiendo de los datos proporcionados por el servidor HTTP desarrollado. En la Figura 4.2-1, se puede observar un ejemplo de este tipo de paneles.

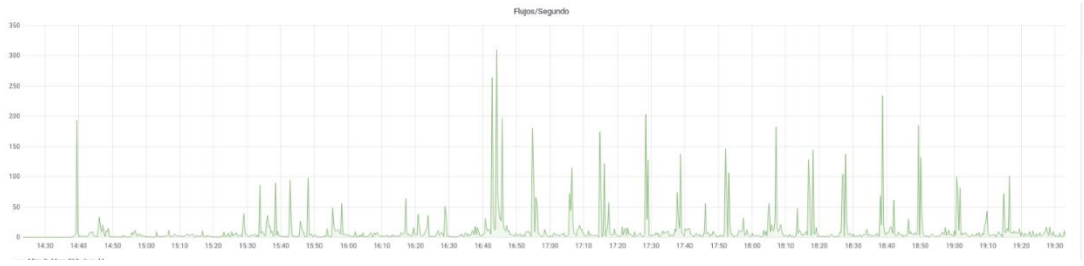

Figura 4.2-1 Ejemplo de gráfico temporal

- **Dato único:** Este tipo de paneles está soportado en la versión de Grafana por defecto, no es necesaria su instalación. Se utiliza para la representación de un dato único. Este panel se puede editar de forma que además de representar el dato único pueda mostrar el porcentaje frente al dato máximo registrado, en forma de semicírculo superior al dato mostrado. También puede mostrar una serie temporal por debajo del propio dato para que el usuario pueda observar la variación del valor en el espacio de tiempo mostrado. Un ejemplo de este tipo de paneles se puede observar en la Figura 4.2-2, que además posee la representación temporal inferior.

Figura 4.2-2 Ejemplo de dato único

- **Texto:** Este tipo de paneles está también soportado en la versión original de Grafana. Como su propio nombre indica, este panel sirve para la representación de un texto fijo en el tablero seleccionado, el panel no realiza petición alguna hacia la fuente de datos a la que esté conectado, de forma que hay que nombrarlo, pero no se le dará mayor importancia en el sistema desarrollado.

- **Mapa de calor:** El mapa de calor es un tipo de panel que también está soportado por defecto en Grafana. Se utiliza para la representación de datos en un mapa de distintos colores que indican visualmente el valor de los campos mostrados sin necesidad de analizarlos uno a uno. Los colores del mapa de calor y sus rangos pueden ser elegidos por el usuario en la pestaña de edición. Un ejemplo de un mapa de calor se puede observar en la Figura 4.2-3, donde se representan los datos en el tipo de colores espectral para facilitar la diferenciación de colores.

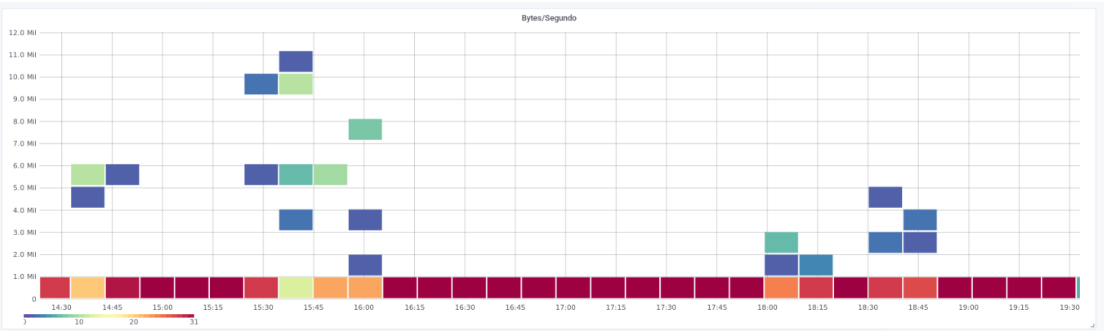

Figura 4.2-3 Ejemplo de mapa de calor

- **Gráfica de tarta:** La gráfica de tarta es un tipo de panel que no viene soportado en la versión por defecto de Grafana. Este panel requiere que el usuario se dirija a la colección de extensiones y proceda a instalarlo. Una vez instalado y reiniciado el servicio de Grafana, la gráfica de tarta permite al usuario la creación de una tarta donde se representan los datos. Cada uno de los datos se pueden activar y desactivar con un click. Esta gráfica de tarta está acompañada por una tabla en la parte inferior,

que permite al usuario tener una visión de los datos como en el panel de una tabla, contenido en la versión por defecto de Grafana. Será de gran utilidad en la representación de direcciones IP, puertos o países de procendencia. Un ejemplo de una gráfica de tarta es el de la Figura 4.2-4.

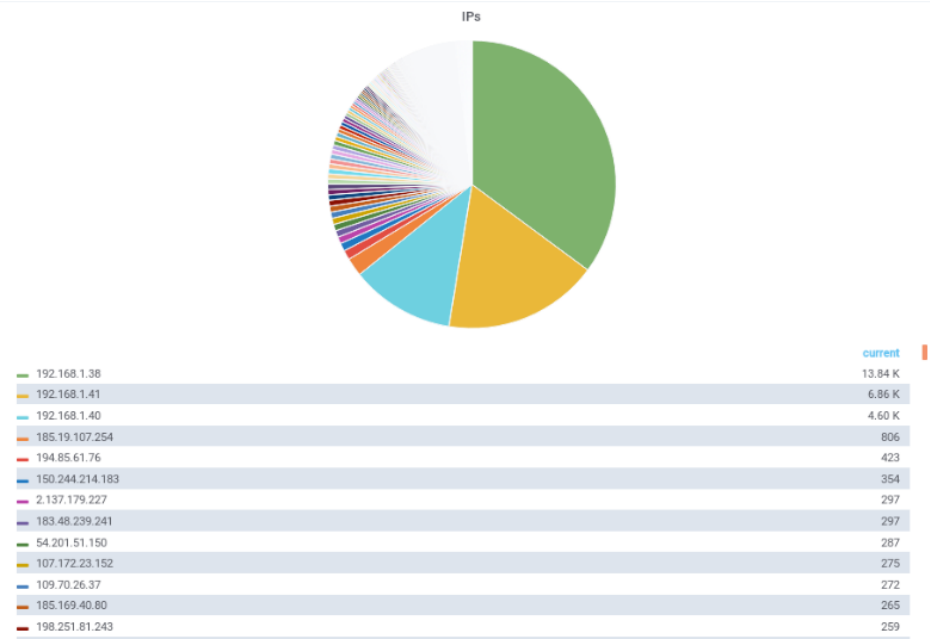

Figura 4.2-4 Ejemplo de gráfica de tarta

- **Mapamundi:** El tipo de panel mapamundi no viene soportado por defecto en Grafana, necesita, al igual que el mapa de calor, que el usuario descargue e instale la extensión. Una vez instalado, el mapamundi nos permite crear un mapa terrestre en el que se pueden representar la posición de los datos reibidos. El mapamundi permite al usuario editar los colores, tamaños y valores de rango de las representaciones. Para el estudio del origen de los paquetes este es, sin duda, el panel más adecuado. En la Figura 4.2-5, se puede observar un ejemplo de este tipo de paneles.

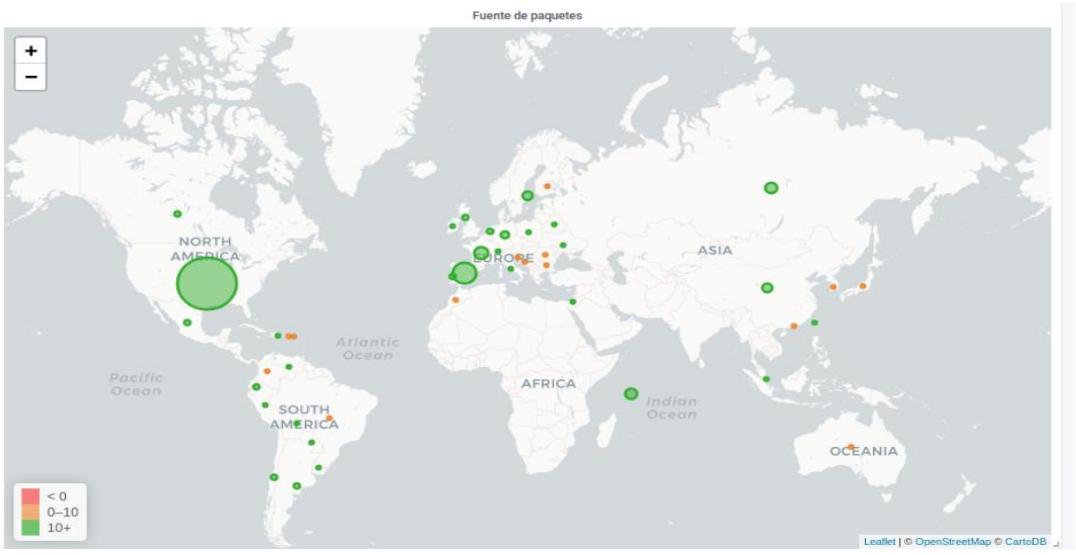

Figura 4.2-5 Ejemplo de mapamundi

Ahora que ya se conocen todos los paneles que están soportados por el servidor HTTP, se procedá a exponer las acciones que lleva a cabo el servidor al recibir las peticiones por parte de Grafana. Al recibir una petición, el servidor realiza el primer paso, transformar la petición desde el formato recibido JSON, a un diccionario de Python. De esta forma, puede trabajar fácilmente con todos los datos recibidos en la petición. Un diccionario de Python es un array que está indexado de forma que todos los datos contenidos en él son fácilmente accesibles en Python. Al obtener el diccionario desde el JSON inicial, el servidor evaluará el contenido de las distintas variables disponibles en el tablero creado, registrará el tiempo en el que se hace la petición y el intervalo de tiempo que el usuario solicita, y por último, el servidor identifica de que tipo de panel proviene la petición.

La diferenciación entre los distintos paneles se realiza a través del máximo de puntos de representación. El principal problema en la utilización de la fuente de datos JSON, reside en que, si se pretende que desde la creación de un panel, este funcione de una forma correcta. Desde el inicio, la única forma que se ha encontrado reside en la utilización de los puntos máximos de representación de los distintos paneles. Cada panel tiene un número distinto de puntos máximos, todos constantes excepto el gráfico temporal, que varía según el tamaño en el tablero. De esta forma el servidor HTTP diferenciará cada panel y construirá una respuesta distinta para cada uno de ellos. Para exponer el funcionamiento de la herramienta en cada caso, el orden de exposición que se seguirá será el mismo que el de la presentación de los distintos paneles.

- **Gráfico temporal:** Tiene un número de puntos variables, por los que este será el caso por defecto del servidor. Al recibir una petición se recuperarán todas las variables y el tiempo en el que se desea filtar.

- **Dato único:** Este tipo de paneles funciona recibiendo todos los datos, igual que funciona un gráfico temporal, de ahí que pueda incluso dibujar un gráfico temporal debajo del dato mostrado. Igual que en el gráfico temporal, se recuperan las variables y el tiempo en el que se desea analizar los datos.

- **Mapa de calor:** El mapa de calor realiza una petición de puntos variables, por lo que entra en el proceso establecido por defecto. En este proceso también se utilizarán las variables previamente almacenadas.

- **Gráfica de tarta:** El panel de las gráficas de tartas también realiza una petición de tres puntos al servidor, por lo que entra en el mismo proceso que el mapa de calor.

- **Mapamundi:** El mapamundi realiza al servidor una petición de un dato. Este será el otro proceso que se podrá ejecutar en el servidor HTTP.

4.2.3 Filtrado de los datos deseados

Una vez que el servidor desarrollado ha dividido las peticiones según el tipo de panel que tiene que construir, procederá a realizar el filtrado de los datos almacenados por SILK, para poder construir una respuesta válida para el panel. A continuación, realizará la comunicación con SILK para efectuar el filtrado. El filtrado se lleva a cabo mediante la utilización de comandos BASH invocados desde el propio servidor

Al dividir los distintos paneles en procesos diferentes, se puede filtrar de forma distinta cada tipo de panel, según sus necesidades. Los paneles de gráficos temporales realizan el mismo filtrado que los paneles de datos únicos, dado que los dos paneles necesitan exactamente los mismos datos para funcionar correctamente. En este caso el servidor realizará una llamada utilizando BASH a la función "filter" de la librería de SILK, que está concatenada a la otra función de SILK mediante el uso de una tubería, la otra función es "rwcount". Estas dos funciones actúan de forma conjunta, "filter" realiza el filtrado de todos los datos almacenados y proporciona todos los datos que cumplen las distintas restricciones de variables y tiempo. Después, estos datos se trasladan como entrada a la función "rwcount" que permite realizar una cuenta del total de paquetes, bytes o flujos que se han registrado en el intervalo de tiempo determinado.

La unión de estas dos potentes herramientas conlleva que el proceso de filtrado y conteo de todos los datos deseados se realice a una velocidad muy elevada, proporcionando datos de retorno rápidamente al servidor.

Los paneles de mapa de calor y de gráficas de tarta no precisan del cuenteo de los datos, sino al contrario, necesitan que se le transmitan todos y cada uno de los datos por separado, para que posteriormente Grafana sea capa de agruparlos para el usuario. De este modo, en el filtrado de los datos solo la realiza la función "filter", obteniendo todos los datos que el usuario desea. Todos los datos obtenidos son comunicados al servidor desarrollado para que pueda utilizarlos.

El panel del mapamundi requiere, al igual que los mapas de calor y las gráficas de tarta, que se le envíen todos los datos que cumplen los requisitos para formar el mapa correctamente. Sin embargo, este tipo de panel necesita además otra información, que los demás no necesitan para posicionar los datos correctamente en el mapa, esto se describirá en el apartado siguiente.

4.2.4 Construcción de la respuesta

Después de tener los datos deseados en Python, el servidor desarrollado continua al paso siguiente, donde se genera un archivo JSON que se transmite a Grafana. Para poder desarrollar esta serie de pasos realizados por el servidor, se analizará cada tipo de panel por separado, dependiendo del proceso en el que se encuentran contenidos. Como se ha descrito, en el servidor existen tres posibilidades distintas, estas posibilidades son:

- La petición se recibe de un **panel de una gráfica temporal o de un panel de dato único**. Estas distintas peticiones comparten el mismo procedimiento, y se lleva a cabo, con los datos extraídos desde las capturas de tráfico de la red, donde se construye un objeto JSON. Se comienza con la cabecera del objeto que es la misma en todos los casos, después se va indexando cada uno de los datos en el objeto. Dependiendo del caso seleccionado se puede responder con los datos de los bytes, los paquetes o los flujos de datos. Una vez se indexan todos los datos en el objeto JSON, el servidor realiza el envío de todos los datos a Grafana, de esta forma se comunica correctamente.

- La petición se recibe de un **panel de mapa de calor o de gráfica de tarta**. Ambas peticiones comparten el mismo método de construcción de la respuesta. Una vez se tienen los datos deseados, el servidor desarrollado decide qué parte de estos datos se transmite a Grafana puesto que existen varias opciones. Estas opciones pueden ser la

dirección IP, el puerto, el país desde el que proviene, o el protocolo en el que se ha realizado. Después de la elección, se crea el objeto que se transmitirá, utilizando una cabecera estándar para todos los casos, e indeando cada dato, uno a uno. Una vez se tienen todos los datos indexados en el objeto JSON, el servidor los transmite a Grafana para que pueda realizar las representaciones oportunas.

- La petición se recibe de un **panel de mapamundi**. Este caso se realiza un proceso único, ya que sus datos previamente filtrados no se pueden indexar en ese formato. Grafana requiere para este tipo de paneles los códigos de los paises, además del número total de datos recibido desde el mismo. Al tener la IP de cada paquete que se procesa, el servidor puede realizar una traducción de la IP al pais de origen de cada uno de los paquetes. Este proceso se realiza mediante el uso del módulo "geoip2.database". De esta forma en el servidor se realiza una suma del total de los datos provenientes de cada pais del mundo y después se continúa creando el objeto JSON que se transmitirá a Grafana, comenzando con la cabecera estándar e indexando posteriormente todos los datos acumulados de la suma del total de cada pais.

4.3 Conclusiones

Para el correcto funcionamiento del sistema diseñado en los capítulos anteriores, en este trabajo se ha prestado especial atención a la creación de un servidor HTTP que habilitará la comunicación entre Grafana y la base de datos formada por YAF y SILK, de tal forma que el sistema finalmente desarrollado funcione de forma óptima.

Con el tipo de respuesta dividida en varias opciones, el servidor HTTP devuelve un objeto distinto en cada momento, dependiendo del tipo de panel del que proviene la petición, siguiendo siempre unos pasos comunes que siempre se llevan a cabo, como sería la recepción de la petición, el análisis de la misma (para discernir entre las distintas opciones), la realización del filtrado con las variables previamente almacenadas y por último la creación de un objeto JSON de acuerdo al tipo de petición, para ser posteriormente remitido a Grafana como respuesta a la petición inicial.

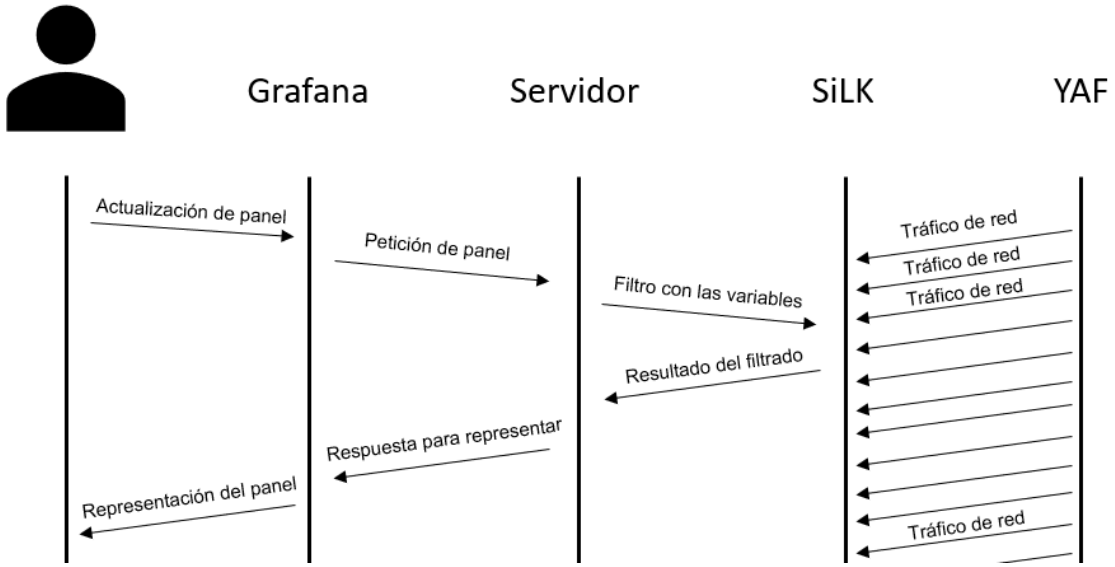

Figura 4.3-1 Esquema de una petición

5 Integración, pruebas y resultados

El objetivo de este capítulo consiste en la integración del sistema desarrollado, que permitirá representar los flujos de la red. Para ello, en este capítulo se expondrán los procedimientos seguidos para integrar el sistema obteniendo el funcionamiento correcto de cada uno de los enlaces, así como, del sistema desarrollado junto con el servidor HTTP creado.

Para proporcionar unos resultados que puedan ser analizados, se han realizado dos pruebas de tiempo, una de respuesta a la herramienta de representación, y otra de espacio de almacenamiento de los datos. Para tener una perspectiva real de los resultados, se compararán con la pila ELK, ya que los sistemas realizan la misma función.

Por último, se realizará una única prueba con una traza obtenida desde Internet, con el fin de poder comprobar que la compresión no se debe a que la IP de destino siempre es la misma, es decir, la IP del PC donde se ha implementado el sistema.

5.1 Extracción de los resultados

Para observar los pasos que se tienen que realizar para ejecutar el sistema desarrollado correctamente, se van a exponer las acciones a llevar a cabo por el usuario para realizarlo. Por facilidad de comprensión, se seguirá el esquema habitual, donde se comienza por la captura y se termina con la respuesta de las peticiones de Grafana.

5.1.1 Captura del tráfico de la red

Una vez se tiene iniciado el PC, lo primero que se debe realizar para ejecutar el sistema desarrollado es activar la recepción de todos los flujos de la red. Para realizar esta acción, y que no se pierdan datos que deberían ser capturados, primero se establece el proceso de escucha de SiLK, de manera que cuando se comience a transmitir con YAF los datos recibidos, todos ellos se almacenarán. La escucha de SiLK se activa mediante el terminal con el siguiente comando BASH:

```
$ /usr/local/sbin/rwflowpack --sensor-conf=/data/sensors.conf --root-dir=/data --
log-directory=/var/log --site-config=/data/SiLK.conf --pack-interfaces
```

Este comando ejecuta el servicio “rwflowpack”, situado en el directorio por defecto, después especifica la dirección del archivo de configuración “sensors.conf”, especifica el directorio donde se desea que se almacenen los datos recibidos, y el directorio donde se realizará el almacenamiento de las acciones del servicio, y por último, especifica el directorio donde se encuentra el archivo de configuración de SILK, “SiLK.conf”, además de decirle que escuche en nuestros interfaces de red disponibles.

Una vez se tiene establecido el proceso de escucha en segundo plano, se procede a comenzar la transmisión de la captura de YAF a SILK. Al igual que pasa con SILK, YAF carece de interfaz gráfica, por lo que este proceso requiere también del uso de un comando en el terminal, el comando utilizado en el sistema es el siguiente:

```
$ yaf --in wlan0 --live pcap --out localhost --ipfix tcp --ipfix-port 18001 --
applabel     --max-payload=500     --SiLK     -d     --log=/var/log/yaf.log     --
pidfile=/var/log/yaf.pid
```

En este comando se hace una invocación al servicio de YAF, en el que se le comunica la interfaz por el que tiene que captar los datos, que se debe de tratar de una captura en tiempo real. Se le aporta la información de dónde tiene que enviar los datos recibidos y que el protocolo de este envío debe ser IPFIX. También se le indica el puerto por el que tiene que redireccionar el tráfico recibido, y el tamaño máximo de los envíos hacia SILK, además de indicarle el directorio donde se almacenará la información de las acciones de YAF, y el directorio en el que debe guardar la información del proceso.

Con estos dos comandos se consigue establecer el sistema de almacenamiento de datos, que se actualiza en tiempo real según todo el tráfico que se recibe de la red. Cabe destacar que ambos comandos necesitan de nivel de administrador en el usuario, es decir, de super-usuario, debido a que, al realizar procesos de escucha y captura en los interfaces o puertos, necesitarán permisos especiales.

Este colector de flujos tiene un funcionamiento independiente al del resto del sistema desarrollado, de manera que si el usuario solo pretende almacenar información que analizará con posterioridad, únicamente debe iniciar la captura de los datos, que se almacenarán de forma automática en el directorio especificado.

5.1.2 Establecimiento del servidor HTTP

Al iniciar el funcionamiento del colector de flujos, todos los datos se irán almacenando automáticamente. Para proceder a la representación de estos, se debe iniciar el servidor HTTP. Para iniciar el servidor, el usuario debe dirigirse al directorio donde esté guardado el mismo, y ejecutarlo con un comando simple de programa de Python. Es importante que el usuario ejecute el servidor HTTP con permisos de administrador, para que todas las acciones que dependen de estos permisos no requieran introducir la contraseña en el terminal.

Al iniciarse el servidor HTTP, se establece la escucha en el puerto especificado en el mismo, que por defecto se ha asignado al puerto 9000, con estos datos hay que continuar con el siguiente paso; el inicio del servicio de servidor de Grafana.

Con todos los pasos anteriores completados, el paso siguiente es dirigirse al terminal del sistema e iniciar el servicio de Grafana, de tal forma que el sistema esté preparado para funcionar correctamente, después, en Grafana, se debe crear la fuente de datos asociada y de esta manera tener la capacidad de generar tableros y paneles con los datos almacenados.

5.2 Pruebas de tiempo

En este apartado se va a someter a varias pruebas la velocidad del sistema desarrollado. Para la realización de este tipo de pruebas, sólo se tendrá en cuanta el tiempo que el sistema tarda desde que Grafana manda la petición al servidor, hasta que esa petición es procesada y la respuesta es recibida por Grafana. Para poder obtener unos resultados ajustados a la realidad este apartado estará divido en dos partes completamente distintas, un subapartado examinará el tiempo cuando la caché está llena y el otro examinará el tiempo que se demora con la caché vacía.

Para poder evaluar la velocidad de respuesta del sistema desarrollado, se analizará también el tiempo de la pila ELK, que tiene la posibilidad gracias a las extensiones de Grafana de sustituir completamente Kibana y utilizar Grafana para la representación, pudiendo así realizar una comparación real entre los resultados de ambos sistemas.

Ambos sistemas cuentan con el mismo tablero, lo que permite reducir el error que se comete en las comparaciones al máximo posible. Sólo se debe cambiar la fuente de datos y seleccionar los deseados para llevar esta acción a cabo. En la Figura 5.2-1 se muestra el tablero con el que se realizarán las pruebas a efectuar.

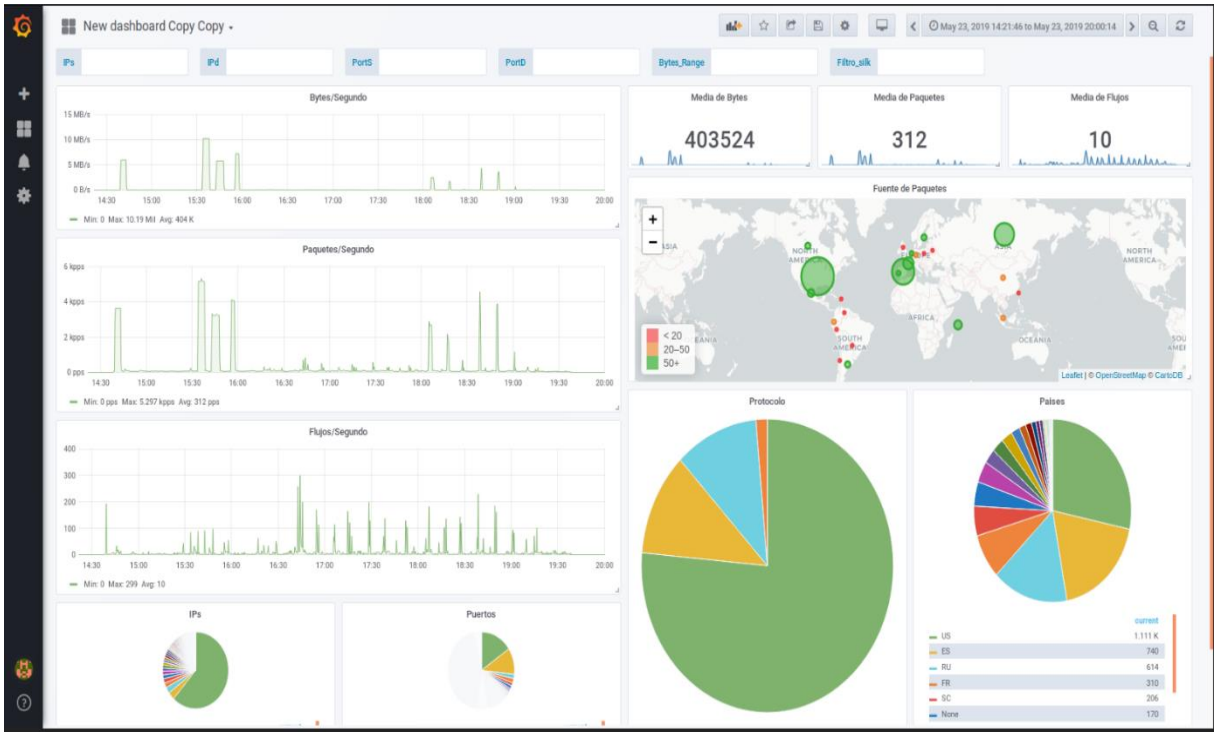

Figura 5.2-1 Tablero de Grafana utilizado en las pruebas

Para la realización de las pruebas se ha creado un tablero desde cero, en el que se han colocado distintos paneles de utilidad para el usuario, como, por ejemplo, las gráficas temporales de los bytes por segundo, los paquetes, o los flujos, también, la media de los resultados como datos simples, un mapamundi del origen de los datos, y por último unas gráficas de tarta para la representación de las IPs, los puertos, las abreviaturas de los países y los protocolos.

En todas estas gráficas se podrá realizar filtrados según las variables que están declaradas en el tablero de Grafana, además de una inmensidad de opciones que la herramienta SILK dispone. Para poder realizar un filtrado de alguna variable que no está declarada en Grafana, se ha habilitado una variable “Filtro_SiLK”, donde se puede introducir el filtro o los filtros separados por un espacio, en el formato especificado en la documentación de SILK.

Hay que hacer notar que estas gráficas son solo una porción de las que se pueden realizar con la herramienta creada, el servidor HTTP. Son las que han parecido las más interesantes de representar en el desarrollo de este TFG. Tan solo se debe variar los campos seleccionados del filtrado de las capturas de tráfico almacenadas por SILK, para realizar la representación de campos totalmente distintos a los mostrados.

Una vez se tiene construido el tablero donde se realizarán las pruebas, no queda más que llenarlo de datos para las mismas. Para realizar una prueba real y equivalente, se decide almacenar en los dos sistemas los mismos datos, teniendo así una igualdad entre ellos y

pudiendo extraer conclusiones. Teniendo en cuenta que ambos sistemas están específicamente diseñados para una cantidad masiva de datos, sería útil explorar esta característica. Por ello, se intentará que la cantidad de datos sea lo más masiva posible. El espacio que ocupa esta captura del tráfico de la red será analizado en las siguientes pruebas.

5.2.1 Prueba con caché llena

En esta prueba, el sistema desarrollado será puesto a prueba con la caché del PC donde se va a realizar la prueba llena. Se llevará a cabo estando en el tablero que se pretende representar y haciendo que recarguen todos los paneles a la vez. En ese momento se medirá el tiempo, hasta que se reciba la respuesta. De esta forma la caché estará totalmente llena con los datos que ha representado previamente en las mismas gráficas.

Con estas premisas, se ejecuta dicho procedimiento y se obtienen los siguientes **resultados** mostrados en la Tabla 5-1:

Tabla 5-1 Resultados de tiempo con caché llena del sistema desarrollado

Nº Prueba	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Tiempo	3,4 s	3,2 s	3,7 s	4,05 s	3,1 s	3,3 s	3,9 s

La media de los resultados es de 3,152 segundos, variando desde un mínimo de 3,1 segundos, hasta el valor máximo de 4,05 segundos. Por sí solos estos valores no aportan mucha información, simplemente, es un tiempo que se puede apreciar a nivel humano y no lo suficientemente grande como para desesperar al usuario.

En este momento, con los resultados obtenidos, el siguiente paso sería realizar exactamente la misma prueba, con los mismos datos en el sistema de la pila ELK. Para ello, como ya se ha expuesto, se utiliza el mismo tablero, cambiando la fuente de datos y ajustando los campos que se van a representar.

Con estas nuevas premisas, se ejecuta el procedimiento anterior con la pila ELK y se obtienen los siguientes **resultados** mostrados en la Tabla 5-2:

Tabla 5-2 Resultados de tiempo con caché llena de la pila ELK

Nº Prueba	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Tiempo	1,2 s	1,3 s	1,1 s	1,05 s	1,15 s	1,35 s	1,2 s

La media de los resultados es de 1,192 segundos, variando desde un mínimo de 1,1 segundos, hasta el valor máximo de 1,35 segundos. Por sí solos estos valores tampoco aportan mucha información, aparte de que son apreciables a nivel humano y que, sin duda, ningún usuario le dará importancia.

Antes de proceder a realizar una comparación entre ambos sistemas hay que destacar que Grafana ha realizado las peticiones de los paneles una a una, es decir, los tiempos mostrados en las tablas siempre serán los de los últimos paneles en cargar, pero en general, la mayoría de los paneles cargan en cascada rápidamente y el usuario siempre puede apreciar que se están representando los datos deseados. Este efecto es bastante más apreciable en el sistema desarrollado, mientras que en la pila ELK hay que prestar atención para observarlo.

Con los resultados obtenidos se pueden sacar conclusiones comparando el sistema diseñado, frente a la pila ELK, al realizar peticiones con la caché llena. Ambos sistemas varían temporalmente entre el máximo y el mínimo. De este modo, y puesto a la amplia variación entre los tiempos puede falsear los resultados si sólo analizamos la media de los resultados, se procede a analizar los mínimos, los máximos y las medias.

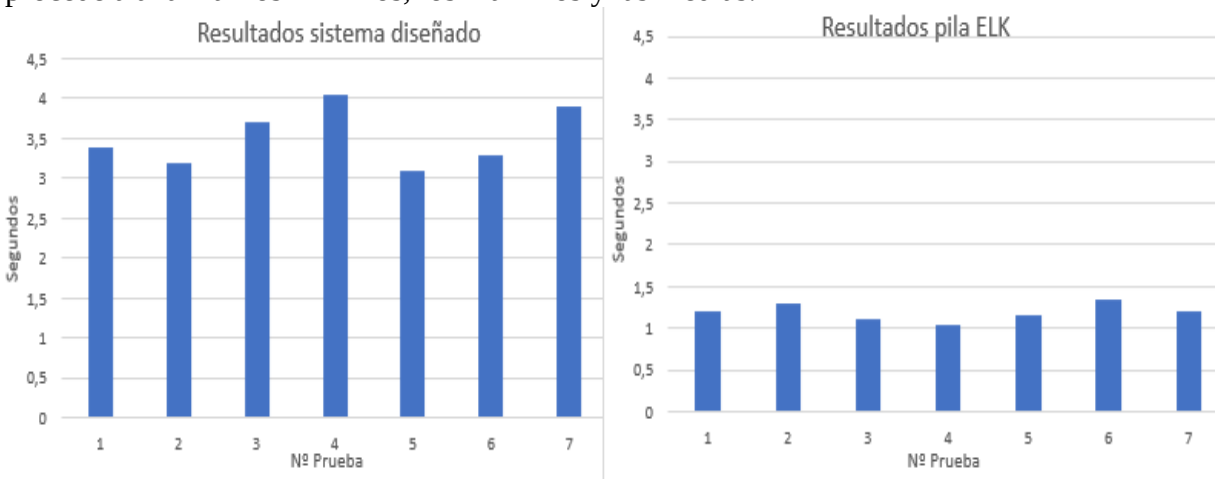

Figura 5.2-2 Tiempo tardado con la caché llena.

- Comenzando por el análisis de los mínimos, se obtiene que el sistema diseñado, es **2,95** veces más lento que la pila ELK.

- Si se continúa analizando las medias, se obtiene que el sistema diseñado es **2,64** veces más lento que la pila ELK.

- Por último, al analizar los máximos de los resultados, se obtiene que el sistema diseñado es **3** veces más lento que la pila ELK.

Estos datos permiten sacar una conclusión evidente, el sistema diseñado es siempre más lento que la pila ELK al tener que realizar transformaciones entre tipos de objetos y además tener que variar en el protocolo de comunicación.

5.2.2 Prueba con caché vacía

Para comprobar la velocidad en otro caso real se ejecutan este tipo de pruebas, es decir, pruebas con la caché vacía. Al realizar estas pruebas se necesita apagar completamente el PC, de modo que la caché se borre totalmente y así quede reiniciado todo el sistema. Al encender de nuevo, se realiza directamente la prueba deseada, ejecutando únicamente lo que sea estrictamente necesario, como en el caso de esta prueba, el inicio del servicio de Grafana y el inicio del servidor HTTP. Una vez realizada la prueba, se vuelve al punto inicial y se repite lo mismo con el otro sistema objeto de análisis, la pila ELK.

Con los mismos datos almacenados y ejecutando el mismo tablero que en las pruebas con la caché llena, se realiza la prueba para el sistema diseñado y se obtienen los **resultados** mostrados en la Tabla 5-3:

Tabla 5-3 Resultados de tiempo con caché vacía del sistema desarrollado

Nº Prueba	**1**	**2**	**3**	**4**	**5**
Tiempo	8,93 s	8,61 s	9,04 s	8,72 s	8,88 s

La media de los resultados es de 8,836 segundos, variando desde un mínimo de 8,61 segundos, hasta el valor máximo de 9,04 segundos. Con estos resultados se puede apreciar la función de la prueba con la caché vacía, el sistema tarda en mostrar los datos más del doble de tiempo, lo que es debido a la concatenación de fallos en caché.

En este momento se procede a realizar el mismo procedimiento en la pila ELK. De estas pruebas se extraen los **resultados** mostrados en la Tabla 5-4:

Tabla 5-4 Resultados de tiempo con caché vacía de la pila ELK

Nº Prueba	**1**	**2**	**3**	**4**	**5**
Tiempo	4,59 s	4,78 s	4,47 s	4,62 s	4,65 s

La media de los resultados es de 4,622 segundos, variando desde un mínimo de 4,47 segundos, hasta el valor máximo de 4,78 segundos. Los resultados de la pila ELK muestran el mismo efecto que en el sistema diseñado, comprobándose en este tipo de pruebas que el tiempo que se tarda en realizar la representación de los datos se multiplica por tres, respecto a las pruebas con la caché llena.

En este caso los resultados temporales son cercanos a la media de los mismos, pero por homogeneidad de presentación de resultados, se procede a exponerlos de la misma forma que en la prueba con la caché totalmente llena.

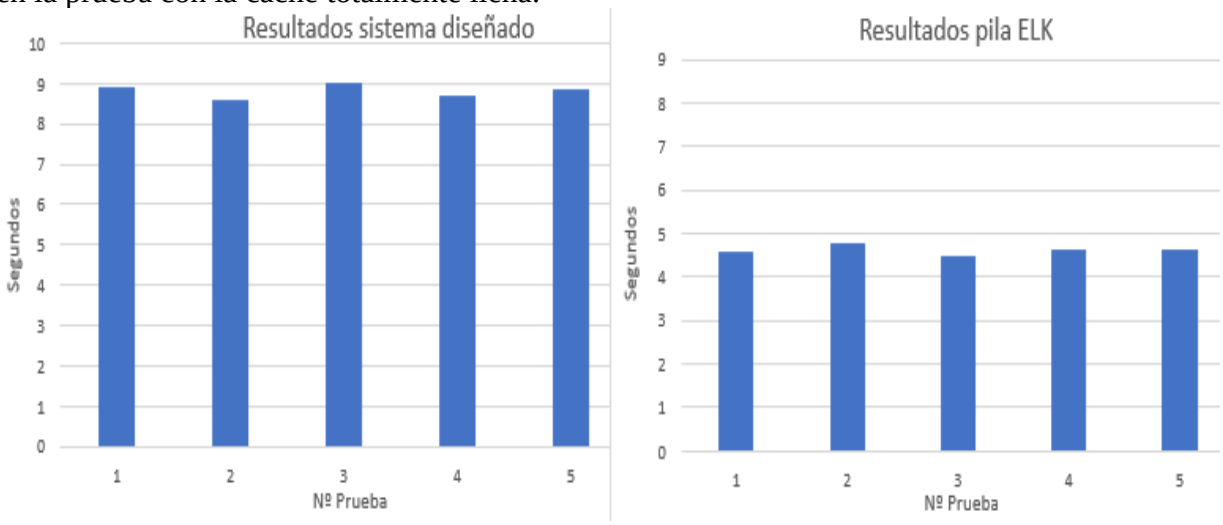

Figura 5.2-3 Tiempo tardado con la caché vacía

- En primer lugar, el análisis de los mínimos arroja que, el sistema diseñado, es **1,92** veces más lento que la pila ELK.

- Continuando el análisis de los resultados de las medias, el sistema diseñado es **1,91** veces más lento que la pila ELK.

  Por último, los resultados de los máximos de las cinco pruebas realizadas arrojan que el sistema diseñado es **1,89** veces más lento que la pila ELK en el peor de los casos.

De esta forma los resultados permiten sacar la conclusión de que, cuando la caché se encuentra vacía, la velocidad del sistema diseñado mejora frente al de la pila ELK. Esto conduce a que, al mostrar datos totalmente nuevos (donde este efecto de caché vacía puede afectar), el sistema diseñado tendrá un rendimiento más cercano al de la pila ELK.

5.3 Prueba de espacio de almacenamiento

En este apartado se prestará especial atención al espacio ocupado por las capturas de tráfico realizadas en ambos sistemas, dado que el sistema diseñado pretende ocupar menos espacio de almacenamiento. La captura de tráfico realizada para esta prueba es la que se utiliza en las pruebas de tiempo. Para tener una cantidad de puntos razonable que permitirán la representación en una gráfica, se ha optado por tomar 20 muestras mientras se realiza la captura. Estas muestras están elegidas por el número de paquetes que contenían, mostrando este número en el eje de abscisas de la Figura 5.3-1.

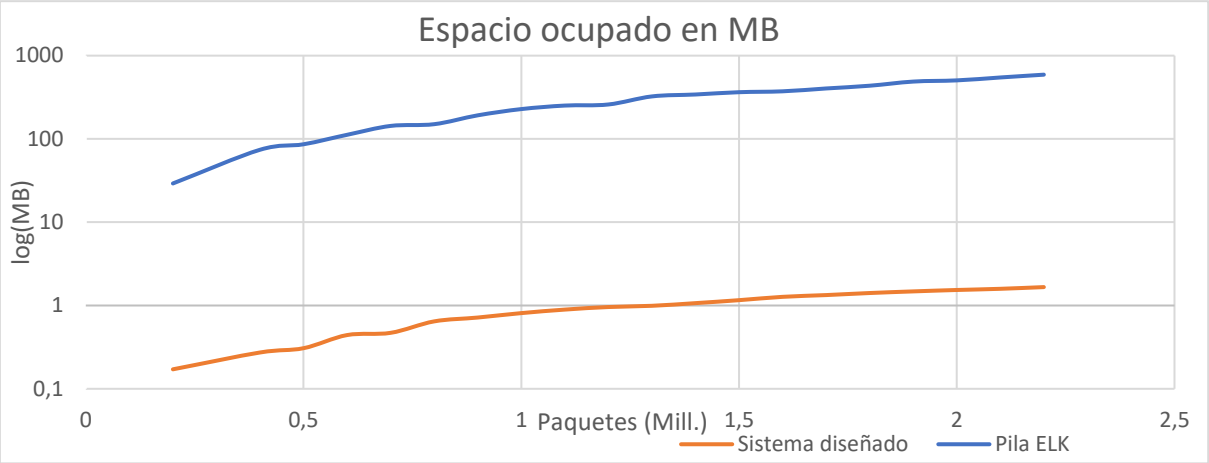

Figura 5.3-1 Tiempo tardado con la caché vacía

En la misma gráfica se puede observar la diferencia existente entre el sistema diseñado y la pila ELK. Tal y como se ha realizado en las pruebas de tiempo, se procede a analizar los resultados obtenidos de los valores de los mínimos, de las medias y de los máximos, para proporcionar al lector la máxima información posible.

- Si se tienen en cuenta los valores mínimos de las diferencias entre el sistema diseñado y la pila ELK se obtiene que el sistema diseñado almacena los datos de forma que ocupan **170,36** veces menos que los datos indexados en la pila ELK.

- En el caso del análisis de la media de las diferencias entre los valores del tamaño de almacenamiento, se obtiene que el sistema diseñado almacena los datos ocupando **291,97** veces menos que los datos indexados en la pila ELK.

- Por último, si se analizan las máximas diferencias entre las cantidades de espacio que ocupan los almacenamientos, el resultado es que, en el sistema diseñado, los datos ocupan **355,04** veces menos que los datos indexados en la pila ELK.

Estos resultados permiten sacar la conclusión de que, al elegir que los datos se guarden en el formato binario de SILK, el tamaño del mismo se reduce significativamente frente al almacenamiento de los mismos en objetos JSON.

La prueba podría haber sido más masiva, pero con esa cantidad de paquetes almacenados, el PC decidió matar el proceso de Elasticsearch, debido a que estaba consumiendo demasiada memoria. En ese momento, se detuvo también la captura en nuestro sistema para estar en igualdad de condiciones.

5.4 Prueba con distintas direcciones IP

Como última comprobación del sistema diseñado, se planteó una posible duda de funcionamiento. ¿Se podría estar dando el caso de que el sistema estuviese ocupando ese espacio tan limitado debido a que al recibir todos los paquetes la misma IP, el formato IPFIX permitiese la reutilización de valores previamente almacenados, para mejorar la compresión de los archivos? Además, la mayoría de los paquetes capturados compartían protocolos y esto también podría utilizarse para comprimir el espacio ocupado todavía más.

Esta prueba se lleva a cabo con archivos PCAP descargados de una la página web referenciada [25], siendo estos unos datos para una competición en el ámbito del análisis de tráfico llamada "Megalodon Challenge" [25]. Este archivo PCAP tiene un tamaño de 2 GB, permitiendo de esta forma, una ingesta masiva de datos en ambos sistemas analizados.

Una vez que la totalidad de los datos están almacenados en ambos sistemas, se procede a analizar el espacio de disco que ocupan, los **resultados** obtenidos, muestran que el espacio ocupado en disco por el sistema diseñado es **257,2** veces menor que el espacio ocupado por los mismos datos almacenados en la pila ELK. De esta forma, se puede observar que la repetición de direcciones IP puede beneficiar al sistema diseñado, pero no es un factor crucial en su diferencia de compresión. En concreto, el archivo PCAP utilizado contaba con 2 669 351 paquetes, unos 400 000 más que el último punto representado en la prueba de espacio.

5.5 Prueba de tiempos de inserción

Cómo prueba adicional, también se han medido los tiempos de inserción de nuevos datos a la base de datos de SiLK frente al tiempo de inserción de Elasticsearch. Para esta prueba se han utilizado los mismos datos para ambos sistemas, de esta forma, se pueden sacar conclusiones. Los resultados se muestran a continuación en la *Tabla* 5-5:

Tabla 5-5 Resultados de tiempo de inserción

Sistema utilizado	Sistema desarrollado	Pila ELK
Segundos	3.0295 s	39.7206 s

Se puede observar que la inserción de los datos tarda **13,11** veces menos en el sistema desarrollado, utilizando YAF y SiLK en vez de la pila ELK.

5.6 Conclusiones

Como conclusión de los resultados, y sin la intención de diferenciar entre el sistema mejor o peor de representación de flujos, se extraen las siguientes conclusiones.

Con los resultados obtenidos se puede apreciar que la velocidad de la pila ELK es bastante mejor que la del sistema diseñado, llegando a ser tres veces mejor en el peor de los casos, pero, sin embargo, la utilización del formato de almacenamiento de registros IPFIX en lugar de utilizar objetos JSON, permite que el sistema diseñado ocupe manos espacio en disco, y de esta forma permita almacenar una mayor cantidad de datos en el mismo espacio.

El principal objetivo del diseño era la reducción del espacio ocupado en disco, y como se puede apreciar por los resultados obtenidos, ese objetivo no se ha perdido en ningún momento del desarrollo, alcanzándose estos resultados finales.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este Trabajo Fin de Grado se ha desarrollado e implementado un servidor HTTP sobre el lenguaje de programación Python, para habilitar el enlace con un sistema de representación de flujos también diseñado. De esta forma se ha conseguido enlazar un sistema totalmente nuevo que comunica un colector de flujo basado en el almacenamiento y la comunicación en formato IPFIX, con la herramienta de representación más popular en la actualidad, Grafana.

Además, se ha demostrado la amplia libertad de Grafana al admitir una amplia variedad de fuentes de datos, demostrando a la vez la capacidad de conectar un servidor HTTP a la fuente de datos JSON.

Con todo lo anterior, se han ejecutado distintas pruebas para la comprobación de la bondad y funcionalidad del sistema, siendo estas pruebas de dos tipos; por un lado, pruebas de tiempos, para poder evaluar la velocidad de respuesta del sistema diseñado y, por otro lado, pruebas de espacio de almacenamiento, para poder evaluar el espacio que ocupan en disco los datos almacenados en la base de datos.

Los resultados de las distintas pruebas efectuadas han permitido la comparación de los mismos, obtenidos con el sistema desarrollado, frente a los obtenidos con el sistema de representación de flujos más popular actualmente, la pila ELK. De esta forma, se ha podido observar la realidad de que el sistema implementado es más lento que la pila ELK, pero la pila ELK ocupa más espacio al almacenar todos los datos capturados en la base de datos, al estar esta indexada en objetos JSON.

En definitiva, la realización del sistema desarrollado, junto con las diversas pruebas efectuadas ha servido como ejemplo de la posibilidad de implementar un enlace entre Grafana y otra herramienta cualquiera, debido a que, al procesar las distintas peticiones, se pueden construir respuestas adecuadas para las mismas, sin importar la procedencia de los datos.

Cabe destacar que la realización de este Trabajo Fin de Grado ha sido una experiencia que ha fomentado el desarrollo de una metodología de trabajo y una mejor administración del tiempo, debido a las constantes fechas de entrega propuestas para la supervisión del tutor. Para poder realizar todas las tareas que se han llevado a cabo ha sido estrictamente necesario el conocimiento adquirido en las asignaturas de Redes a lo largo de la carrera, tales como Arquitectura de Redes I, Arquitectura de Redes II, Redes Multimedia, Sistemas Distribuidos, Fundamentos de Sistemas Informáticos, Programación I o Programación II.

Todas estas asignaturas han aportado la base de los conocimientos necesarios para afrontar un proyecto en el ámbito de las Redes, siendo necesarios conocimientos sobre todos los protocolos utilizados, como por ejemplo HTTP o TCP. También ha sido necesario el conocimiento de los objetos JSON o el protocolo IPFIX. Y junto con el aprendizaje sobre los lenguajes de programación Python y BASH, se ha podido completar la realización del proyecto completo.

Además, ha sido necesario la investigación y comprensión de todas las herramientas utilizadas para el sistema diseñado, y su competidor en las pruebas, la pila ELK. Se han

adquirido conocimientos principalmente de las herramientas YAF, SILK y Grafana, habilitando de esta manera explotar al máximo posible todas sus características y funciones.

Por último, cabe destacar que el trabajo está publicado en la página web "GitHub.com" [26], estando accesible públicamente desde la siguiente dirección URL: https://github.com/RaulParis/Grafana_and_SiLK_connection_using_a_Python_HTTP_Server [27]

O, por medio de código QR:

Figura 6.1-1 QR de la publicación en GitHub

También se ha llevado a cabo la resolución de *bugs* encontrados en Grafana, como: tiempo en UTC y "muerte" de paneles. Además de la creación de código alternativo para comprobar si había posibilidad de mejora. Ejemplos: lectura del JSON de SiLK o creación de uno en Python, variación de la lectura de los datos directamente leyéndolos desde Python. Y la creación de dos *dashboards* distintos, uno mostrando todas las gráficas a la vez y otro pudiendo elegir con un panel de cada tipo.

6.2 Trabajo futuro

Todo el trabajo realizado puede servir de punto de partida para otros trabajos en un futuro, puesto que el sistema diseñado y desarrollado es mejor en compresión de datos, pero es peor, en la velocidad de respuesta de peticiones. Además, se abre una gran variedad de posibilidades que pueden ser exploradas en un futuro como, por ejemplo:

- **Mejora de velocidad:** Este posible punto de partida podría servir para la mejora del sistema completo, de modo que la velocidad de respuesta pueda aumentar considerablemente, esta mejora se podría llevar a cabo con un cambio en el lenguaje de programación utilizado en el servidor HTTP, pasando a utilizar C como lenguaje para su creación.

- **Conexión con otras herramientas:** Ha quedado demostrada la posibilidad de que Grafana puede conectarse con herramientas que ni siquiera conoce, de esta forma, con los conocimientos adquiridos en este Trabajo Fin de Grado, se podrían implementar otros diseños de representación de flujos enlazando Grafana con un sinfín de herramientas distintas.

- **Creación de una herramienta de representación:** Durante la realización de todo el Trabajo Fin de Grado, se han capturado peticiones desde Grafana hacia el servidor HTTP, de forma que se conocen los campos necesarios para la representación de los distintos paneles. En este punto de partida se puede plantear la creación de una nueva herramienta de representación, que reciba los datos de igual forma que lo hace Grafana.

Por todo ello este Trabajo Fin de Grado puede utilizarse de punto de partida para otros posibles trabajos finales de grado en un futuro.

Referencias

[1] «Internet usage statistics,» Marzo 2019. [En línea]. Available: https://www.internetworldstats.com/stats.htm.

[2] C. Castelló Llantada, «Las tecnológicas estadounidenses lideran de nuevo la lista de las empresas más grandes del mundo,» 2 Enero 2019. [En línea]. Available: https://cincodias.elpais.com/cincodias/2018/12/28/companias/1546023529_428376.html.

[3] M. G. Vaquero, Desarrollo de un sistema de monotorización de red basado en tecnologías de tratamiento masivo de datos, Madrid: UAM, 2018.

[4] E. B. L. M. T. Z. A. W. B. Trammell, «Specification of the IP Flow Information Export (IPFIX) File Format,» Octubre 2009. [En línea]. Available: https://tools.ietf.org/html/rfc5655.

[5] tcpdump, «PCAP,» 30 Agosto 2004. [En línea]. Available: https://www.tcpdump.org/pcap/pcap.html.

[6] E. T. Bray, «The JavaScript Object Notation (JSON) Data Interchange Format,» [En línea]. Available: https://tools.ietf.org/html/rfc7159.

[7] «Python,» [En línea]. Available: https://www.python.org/.

[8] I. SolidMatrix Technologies y S. Y., «Common Format and MIME Type for Comma-Separated Values (CSV) Files,» Octubre 2005. [En línea]. Available: https://tools.ietf.org/html/rfc4180.

[9] Solarwinds, «Solarwinds,» [En línea]. Available: https://www.solarwinds.com/es/netflow-traffic-analyzer.

[10] «Wireshark,» [En línea]. Available: https://www.wireshark.org/.

[11] Plixer, «Scrutinizer,» [En línea]. Available: https://www.plixer.com/products/scrutinizer/.

[12] «ELK STACK,» [En línea]. Available: https://www.elastic.co/es/.

[13] CERT NetSA Security Suite, «YAF,» [En línea]. Available: https://tools.netsa.cert.org/yaf/index.html.

[14] CERT NetSA Security Suite, «SILK,» [En línea]. Available: https://tools.netsa.cert.org/SiLK/index.html.

[15] Pluralsight, «Javascript,» [En línea]. Available: https://www.javascript.com/.

[16] J. Postel, «Internet Protocol,» [En línea]. Available: https://tools.ietf.org/html/rfc791.

[17] «Grafana,» [En línea]. Available: https://grafana.com/.

[18] GNU, «GNU Bash,» [En línea]. Available: https://www.gnu.org/software/bash/.

[19] F. R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. B.-L. , «Hypertext Transfer Protocol,» Junio 1999. [En línea]. Available: https://tools.ietf.org/html/rfc2616.

[20] ANSI, «ISO/IEC 9899:2011,» [En línea]. Available: https://webstore.ansi.org/Standards/ISO/ISOIEC98992011.

[21] Simpod, «JSON Datasource,» 2019. [En línea]. Available: https://grafana.com/plugins/simpod-json-datasource.

[22] Grafana Labs, «SimpleJSON Datasource,» [En línea]. Available: https://grafana.com/plugins/grafana-simple-json-datasource.

[23] J. Postel, «Transmission control protocol,» Septiembre 1981. [En línea]. Available: https://tools.ietf.org/html/rfc793.

[24] D. Connolly y T. Berners-Lee, «Hypertext Markup Language,» [En línea]. Available: https://tools.ietf.org/html/rfc1866.

[25] Packet-Foo, «Megalodon Challenge,» 26 Julio 2015. [En línea]. Available: https://blog.packet-foo.com/2015/07/the-megalodon-challenge/.

[26] «GitHub,» [En línea]. Available: https://github.com.

[27] R. P. Murillo, «Grafana_and_SiLK_connection_using_a_Python_HTTP_Server,» 12 Junio 2019. [En línea]. Available: https://github.com/RaulParis/Grafana_and_SiLK_connection_using_a_Python_HTTP_Server.

[28] R. Khanna, «How to use Elasticsearch, Logstash and Kibana to visualise logs in Python in realtime,» [En línea]. Available: https://www.freecodecamp.org/news/how-to-use-elasticsearch-logstash-and-kibana-to-visualise-logs-in-python-in-realtime-acaab281c9de/.

[29] «The ELK STACK,» [En línea]. Available: https://cr4ft-ing.tistory.com/189.

[30] funkboje, «Solaranzeige, InfluxDB und Grafana,» [En línea]. Available: https://www.photovoltaikforum.com/thread/125059-solaranzeige-influxdb-und-grafana/.

Anexos

A. Fichero de configuración de sensores

Al configurar el colector de flujo, es necesaria la creación de un fichero de configuración de sensores. En este fichero se crea el sensor indicándole el puerto en el que va a escuchar, el protocolo utilizado y la IP donde se realizará la escucha. Después se define el grupo dentro del la red local y por último se asigna el sensor.

En el caso del sistema desarrollado, se ha respetado el puerto establecidos por defecto, para facilitar la utilización de las herramientas.

Este archivo ha sido llamado “sensors.conf” y se muestra en la Figura A-1.

```
1. probe S0 ipfix
2.  listen-on-port 18001
3.  protocol tcp
4.  listen-as-host 127.0.0.1
5. end probe
6.
7. group my-network
8.  ipblocks 192.168.1.0/24 # address of eth0.
9.  ipblocks 10.0.0.0/8 # other blocks you consider internal
10.  end group
11.
12.  sensor S0
13.   ipfix-probes S0
14.   internal-ipblocks @my-network
15.   external-ipblocks remainder
16.  end sensor
```

Figura A-1 Archivo de configuración de sensores

B. Creación del datasource en Grafana

Para realizar la instalación de la fuente de datos JSON, es necesario dirigirse a la página oficial de Grafana, en ella hay que desplazarse al apartado *Plugins*. En este apartado, se pueden observar todas las extensiones que Grafana tiene disponibles, para el caso de una fuente de datos, hay que seleccionar *Data Sources*. Una vez la opción está seleccionada, tan sólo es necesario desplazarse hacia abajo para encontrar la fuente de datos deseada, "JSON".

Instalar la fuente es sencillo, tan sólo es necesario continuar los pasos que el mismo desarrollador proporciona. Una vez instalada, es necesario el reinicio del servicio de Grafana para el correcto funcionamiento de la extensión instalada.

Después de reiniciar Grafana, hay que acceder al apartado de *Data Sources* en las opciones de configuración, allí se puede crear una nueva fuente de datos, utilizando la extensión previamente descargada. Como configuración de ejemplo se muestra la utilizada en el sistema desarrollado, expuesta en la Figura B-1.

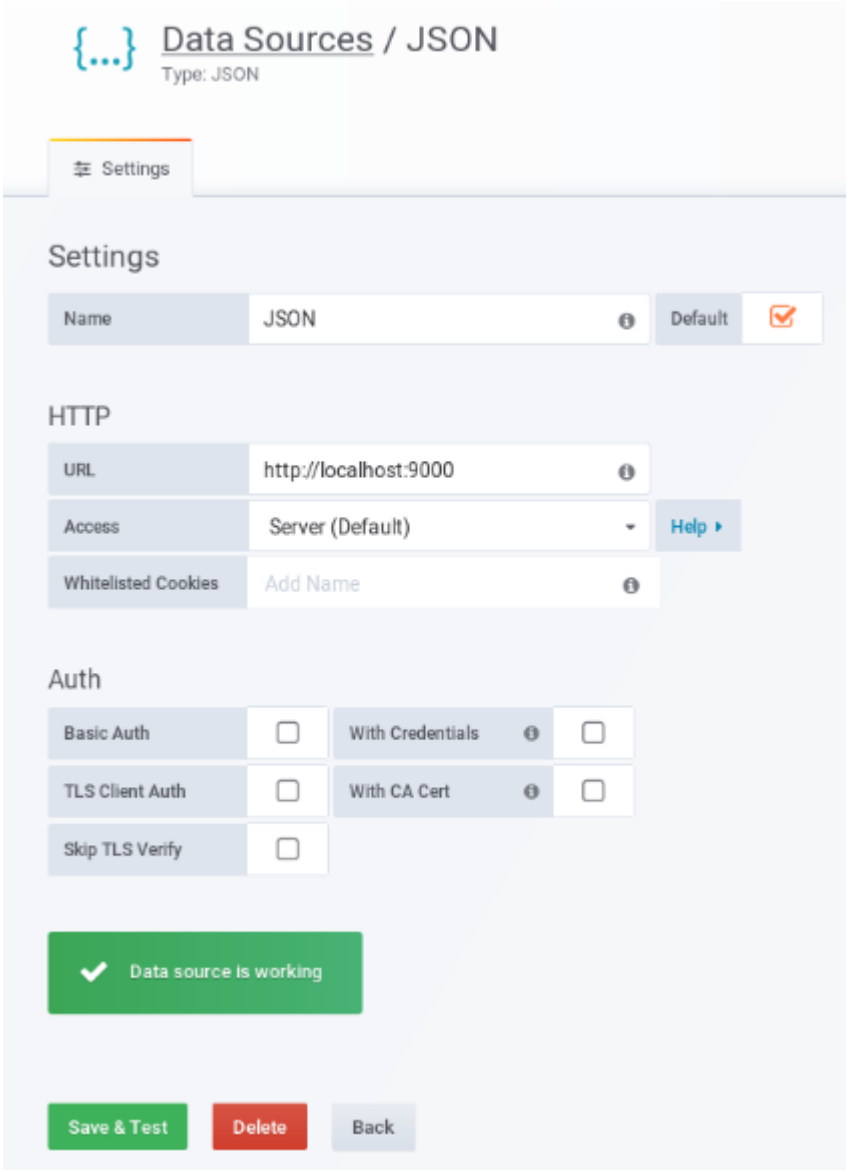

Figura B-1 Creación de fuente de da

C. Petición de Grafana

A lo largo de la implementación del diseño desarrollado se ha hecho alusión a las peticiones que realiza Grafana al servidor HTTP, estas peticiones varían dependiendo del tipo de panel que se ha creado o de la fuente de datos utilizada.

En el caso del sistema desarrollado, se ha utilizado la fuente de datos JSON, que habilita las comunicaciones por medio de la utilización de objetos JSON como petición y respuesta a Grafana, de esta forma se ha habilitado las comunicaciones entre las distintas partes del sistema desarrollado.

Como ejemplo de una petición de Grafana, se ha recopilado una petición de Grafana realizada hacía el servidor HTTP para un panel de gráfica temporal. En la Figura C-1, se pueden apreciar todos los campos analizados y desarrollados en la realización de este Trabajo Fin de Grado. Para facilitar la comprensión se ha decidido exponer en el ejemplo el objeto JSON descifrado por el servidor HTTP, puesto que la petición real se recibe en formato de tráfico de red, es decir, en bytes.

```
1. {'timezone': 'browser',
2.  'panelId': 28,
3.  'dashboardId': 2,
4.  'range': {
5.           'from': '2019-05-23T12:27:02.978Z',
6.           'to': '2019-05-23T17:11:54.842Z',
7.           'raw': {
8.                      'from': '2019-05-23T12:27:02.978Z',
9.                      'to': '2019-05-23T17:11:54.842Z'
10.                     }
11.            },
12.   'rangeRaw': {
13.                'from': '2019-05-23T12:27:02.978Z',
14.                'to': '2019-05-23T17:11:54.842Z'},
15.                'interval': '20s',
16.                'intervalMs': 20000,
17.                'targets': [
18.                           {'data': None}
19.                          ],
20.                'maxDataPoints': 927,
21.                'scopedVars': {
22.                                'IPs': {
23.                                        'text': '',
24.                                        'value': ''
25.                                       },
26.                                'IPd': {
27.                                        'text': '',
28.                                        'value': ''
29.                                       },
30.                                'PortS': {
31.                                          'text': '',
32.                                          'value': ''
33.                                         },
34.                                'PortD': {
35.                                          'text': '',
36.                                          'value': ''
37.                                         },
38.                                'Bytes_Range': {
```

```
                                                'text': '',
                                                'value': ''
                                            },
                                    'Filtro_SiLK': {
                                                'text': '',
                                                'value': ''
                                               },
                                    '__interval': {
                                                'text': '20s',
                                                'value': '20s'
                                               },
                                    '__interval_ms': {
                                                 'text': 20000,
                                                'value':20000
                                                }
                                },
    'adhocFilters': []
    }
```

Figura C-1 Ejemplo de petición de Grafana

D. Servidor HTTP

El código utilizado en el desarrollo del servidor HTTP basado en Python, se muestra a continuación, en la Figura D-1.

```
1. from __future__ import print_function
2. from http.server import BaseHTTPRequestHandler, HTTPServer
3. from SiLK import *
4. import time
5. import json
6. import sys
7. import os
8. import datetime
9. import geoip2.database
10.  from collections import defaultdict
11.
12.  #Definimos las variables globales que serán utilizadas a lo largo
   del código
13.  hostName = "localhost"
14.  hostPort = 9000
15.  #Esta es la base de datos del módulo geoip2, descargada de su
   página oficial
16.  reader = geoip2.database.Reader('/data/prueba2/GeoLite2-
   City_20190319/GeoLite2-City.mmdb')
17.  auxi1=1
18.  auxi2=1
19.  auxi3=1
20.  auxi4=1
21.  global time_minmej
22.  global time_maxmej
23.  time_minmej=1
24.  time_maxmej=1
25.
26.  #Definimos el servidor
27.  class MyServer(BaseHTTPRequestHandler):
28.  def do_GET(self): #Si es una petición GET
29.  self.send_response(200) #Repondemos 200
30.  self.send_header("Content-type", "text/html") #Tipo HTML
31.  self.end_headers()
32.  def do_POST(self): #Si es una petición POST
33.  content_type = (self.headers.get('content-type',0)) #Verificar json
34.  global values #Definimos values como global
35.  if (content_type=="application/json"): #Si se trata de un objeto
   JSON
36.  if (self.path != "/search"): #Y viene dirigido a /search
37.  content_len = int(self.headers.get('content-length',0)) #Verificar
   json
38.
39.  post_body = self.rfile.read(content_len)
40.
41.  body_aux = json.loads(post_body)
42.
43.  time_min = (body_aux['range'])['from'] #Extraemos el tiempo mínimo
   de la petición
44.  time_min = datetime.datetime.strptime(time_min, "%Y-%m-
   %dT%H:%M:%S.%fZ") # Lo convertimos a datetime
45.  time_min = time_min + datetime.timedelta(hours=0)
46.  time_min_aux =
   datetime.datetime(time_min.year,time_min.month,time_min.day,time_min.
   hour,time_min.minute,time_min.second).timestamp()
```

```
47.  time_min_aux =
     datetime.datetime(time_min.year,time_min.month,time_min.day,time_min.
     hour,time_min.minute,time_min.second).replace(tzinfo=datetime.timezon
     e.utc).timestamp() #Hacemos cambio al tiempo con nuestra hora local
48.
49.  time_max = (body_aux['range'])['to'] #Extraemos el tiempo mínimo de
     la petición
50.  time_max = datetime.datetime.strptime(time_max, "%Y-%m-
     %dT%H:%M:%S.%fZ") # Lo convertimos a datetime
51.  time_max = time_max + datetime.timedelta(hours=0)
52.  time_max_aux =
     datetime.datetime(time_max.year,time_max.month,time_max.day,time_max.
     hour,time_max.minute,time_max.second).timestamp()
53.  time_max_aux =
     datetime.datetime(time_max.year,time_max.month,time_max.day,time_max.
     hour,time_max.minute,time_max.second).replace(tzinfo=datetime.timezon
     e.utc).timestamp() #Hacemos cambio al tiempo con nuestra hora local
54.
55.  interval = body_aux['intervalMs']/1000
56.
57.
58.  variables = "" #Creamos la variable donde se almacenarán las
     variables de Grafana
59.
60.  if 'IPs' in (body_aux['scopedVars']): #Si existe el campo en
     scopedVars
61.  if (((body_aux['scopedVars'])['IPs'])['text']) != "": #Si contiene
     algo
62.  variables = variables + " --scidr=" +
     str(((body_aux['scopedVars'])['IPs'])['text'])
63.
64.  if 'IPd' in (body_aux['scopedVars']): #Si existe el campo en
     scopedVars
65.  if (((body_aux['scopedVars'])['IPd'])['text']) != "": #Si contiene
     algo
66.  variables = variables + " --dcidr=" +
     str(((body_aux['scopedVars'])['IPd'])['text'])
67.
68.  if 'PortS' in (body_aux['scopedVars']): #Si existe el campo en
     scopedVars
69.  if (((body_aux['scopedVars'])['PortS'])['text']) != "": #Si
     contiene algo
70.  variables = variables + " --sport=" +
     str(((body_aux['scopedVars'])['PortS'])['text'])
71.
72.  if 'PortD' in (body_aux['scopedVars']): #Si existe el campo en
     scopedVars
73.  if (((body_aux['scopedVars'])['PortD'])['text']) != "": #Si
     contiene algo
74.  variables = variables + " --dport=" +
     str(((body_aux['scopedVars'])['PortD'])['text'])
75.
76.  if 'Bytes_Range' in (body_aux['scopedVars']): #Si existe el campo
     en scopedVars
77.  if (((body_aux['scopedVars'])['Bytes_Range'])['text']) != "": #Si
     contiene algo
78.  variables = variables + " --bytes=" +
     str(((body_aux['scopedVars'])['Bytes_Range'])['text'])
79.
80.  if 'Filtro_SiLK' in (body_aux['scopedVars']): #Si existe el campo
     en scopedVars
```

```
81.  if (((body_aux['scopedVars'])['Filtro_SiLK'])['text']) != "": #Si
     contiene algo
82.  variables = variables + " " +
     str(((body_aux['scopedVars'])['Filtro_SiLK'])['text'])
83.
84.  if (body_aux['maxDataPoints']) == 1: #En el caso de que
     maxdatapoints sea 1
85.  global time_minmej #Cargamos variables globales dentro del if
86.  global time_maxmej
87.  if (time_min != time_minmej) or (time_max != time_maxmej): #Si la
     petición no tiene el mismo rango de tiempo que la anterior
88.  try:
89.  os.system('rm filtro.rw') #Se borra el anterior
90.  #Y se realiza el nuevo filtro
91.  comando = 'sudo rwfilter --start-
     date='+str(time_min.year)+'/'+str(time_min.month)+'/'+str(time_min.da
     y)+':'+str(time_min.hour)+' --end-
     date='+str(time_max.year)+'/'+str(time_max.month)+'/'+str(time_max.da
     y)+':'+str(time_max.hour)+' --type=all ' + str(variables) + ' --
     proto=6 --pass-destination=filtro.rw'
92.  os.system(comando)
93.
94.  except:
95.  hostPort = 9000
96.  time_maxmej = time_max #Guardamos los rangos de tiempo de las
     peticiones
97.  time_minmej = time_min
98.
99.  infile = SiLKfile_open("filtro.rw", READ) #Leemos el resultado del
     filtro
100.
101. values = '[' #Creamos la variable de la respuesta
102. my_dict = defaultdict(int) #Inicializamos un diccionario de ints
103.
104. for rec in infile: #Para cada registro del archivo de registros
105. try:
106. response = reader.city(str(rec.sip)) #Localizamos la IP
107. my_dict[response.country.iso_code] += 1 #La indexamos en el
     diccionario
108. except:
109. hostPort = 9000
110. for key in my_dict: #Para cada clave del diccionario
111. values = values + '{"target": ' + '"' + str(key) + '",
     "datapoints": [[' + str(my_dict[key]) + ',' +
     str(rec.etime_epoch_secs) + ']]},' #Construimos la respuesta
112. values = values[:-1] + "]" #Cerramos el JSON
113.
114. elif (body_aux['maxDataPoints']) == 3: #En el caso de que
     maxdatapoints sea 3
115.
116. if (time_min != time_minmej) or (time_max != time_maxmej): #Si la
     petición no tiene el mismo rango de tiempo que la anterior
117. try:
118. os.system('rm filtro.rw') #Se borra el anterior
119. #Y se realiza el nuevo filtro
120. comando = 'sudo rwfilter --start-
     date='+str(time_min.year)+'/'+str(time_min.month)+'/'+str(time_min.da
     y)+':'+str(time_min.hour)+' --end-
     date='+str(time_max.year)+'/'+str(time_max.month)+'/'+str(time_max.da
     y)+':'+str(time_max.hour)+' --type=all ' + str(variables) + ' --
     proto=6 --pass-destination=filtro.rw'
```

```
121. os.system(comando)
122.
123. except:
124. hostPort = 9000
125. time_maxmej = time_max #Guardamos los rangos de tiempo de las
   peticiones
126. time_minmej = time_min
127.
128. infile = SiLKfile_open("filtro.rw", READ) #Leemos el resultado del
   filtro
129.
130. global auxi4
131.
132. seleccion_aux_pie = (auxi4%4)+1 #Calculamos la respresenación
133.
134. if (seleccion_aux_pie) == 1: #Si es 1
135. seleccion_aux_pie = 'CountryS' #Se representarán países
136.
137. elif (seleccion_aux_pie) == 2: #Si es 2
138. seleccion_aux_pie = 'IPS' #Se representarán IPs
139.
140. elif (seleccion_aux_pie) == 3: #Si es 3
141. seleccion_aux_pie = 'PortS' #Se representarán puertos
142.
143. else: #Sino
144. seleccion_aux_pie = 'application' #Se representará la aplicación
145.
146. auxi4 +=1       #Sumamos 1 a la variable
147. values = '[' #Creamos la variable de la respuesta
148. my_dict = defaultdict(int) #Inicializamos un diccionario de ints
149.
150. if (seleccion_aux_pie) == 'CountryS': #Si se representarán países
151.
152. for rec in infile: #Para cada registro del archivo de registros
153. try:
154. response = reader.city(str(rec.sip)) #Localizamos la IP
155. my_dict[response.country.iso_code] += 1 #La indexamos en el
   diccionario
156. except:
157. hostPort = 9000
158. for key, value in sorted(my_dict.items(), key=lambda x: x[1],
   reverse=True): #Para cada clave del diccionario ordenado de menor a
   mayor
159. values = values + '{"target": ' + '"' + str(key) + '",
   "datapoints": [[' + str(value) + ',' + str(rec.etime_epoch_secs) +
   ']]},' #Indexamos los valores
160. elif (seleccion_aux_pie) == 'IPS': #Si se representarán IPs
161.
162. for rec in infile: #Para cada registro del archivo de registros
163. try:
164. my_dict[rec.sip] += 1 #La indexamos en el diccionario
165. except:
166. hostPort = 9000
167. for key, value in sorted(my_dict.items(), key=lambda x: x[1],
   reverse=True): #Para cada clave del diccionario ordenado de menor a
   mayor
168. values = values + '{"target": ' + '"' + str(key) + '",
   "datapoints": [[' + str(value) + ',' + str(rec.etime_epoch_secs) +
   ']]},' #Indexamos los valores
169.
170. elif (seleccion_aux_pie) == 'PortS': #Si se representarán puertos
```

```
171.
172. for rec in infile: #Para cada registro del archivo de registros
173. try:
174. my_dict[rec.sport] += 1 #La indexamos en el diccionario
175. except:
176. hostPort = 9000
177. for key, value in sorted(my_dict.items(), key=lambda x: x[1],
   reverse=True): #Para cada clave del diccionario ordenado de menor a
   mayor
178. values = values + '{"target": ' + '"' + str(key) + '",
   "datapoints": [[' + str(value) + ',' + str(rec.etime_epoch_secs) +
   ']]},' #Indexamos los valores
179.
180. elif (seleccion_aux_pie) == 'application': #Si se representarán
   aplicaciones
181.
182. for rec in infile: #Para cada registro del archivo de registros
183. try:
184. my_dict[rec.application] += 1 #La indexamos en el diccionario
185. except:
186. hostPort = 9000
187. for key, value in sorted(my_dict.items(), key=lambda x: x[1],
   reverse=True): #Para cada clave del diccionario ordenado de menor a
   mayor
188. values = values + '{"target": ' + '"' + str(key) + '",
   "datapoints": [[' + str(value) + ',' + str(rec.etime_epoch_secs) +
   ']]},' #Indexamos los valores
189.
190. values = values[:-1] + "]" #Cerramos el objeto JSON
191.
192. else: #Si no tiene ni 1 ni 3 en el maxdatopints
193.
194. if 'target' in (body_aux['targets']): #SI existe el campo
195. filters = ((body_aux['targets'])['target']) #Se guarda en filters
196. #Se realiza el filtro concatenado con el cuenteo
197. comando = 'sudo rwfilter --start-
   date='+str(time_min.year)+'/'+str(time_min.month)+'/'+str(time_min.da
   y)+':'+str(time_min.hour)+' --end-
   date='+str(time_max.year)+'/'+str(time_max.month)+'/'+str(time_max.da
   y)+':'+str(time_max.hour)+' --type=all ' + str(filters) +
   str(variables) + ' --proto=6 --pass=stdout | rwcount --start-time=' +
   str(time_min.year) + '/' + str(time_min.month) + '/' +
   str(time_min.day) + ':'  +  str(time_min.hour) + ':' +
   str(time_min.minute) + ':' + str(time_min.second) + ' --end-time=' +
   str(time_max.year) + '/' + str(time_max.month) + '/' +
   str(time_max.day) + ':'  +  str(time_max.hour) + ':' +
   str(time_max.minute) + ':' + str(time_max.second) + ' --bin-size=' +
   str(interval) + '>cuenta.txt'
198.
199. else: #Si no existen variables
200. comando = 'sudo rwfilter --start-
   date='+str(time_min.year)+'/'+str(time_min.month)+'/'+str(time_min.da
   y)+':'+str(time_min.hour)+' --end-
   date='+str(time_max.year)+'/'+str(time_max.month)+'/'+str(time_max.da
   y)+':'+str(time_max.hour)+' --type=all ' + str(variables) + ' --
   proto=6 --pass=stdout | rwcount --start-time=' + str(time_min.year) +
   '/' + str(time_min.month) + '/' + str(time_min.day) + ':'  +
   str(time_min.hour) + ':' + str(time_min.minute) + ':' +
   str(time_min.second) + ' --end-time=' + str(time_max.year) + '/' +
   str(time_max.month) + '/' + str(time_max.day) + ':'  +
```

```
   str(time_max.hour) + ':' + str(time_max.minute) + ':' +
   str(time_max.second) + ' --bin-size=' + str(interval) + '>cuenta.txt'
201.
202. os.system(comando) #Se ejecuta
203.
204. if (body_aux['maxDataPoints']) == 1: #En el caso de que
   maxdatapoints sea 1
205. global auxi1 #Se invoca la variable
206. seleccion_aux = (auxi1%3)+1 #Se incrementa en 1
207. auxi1 += 1 #Se incrementa en 1
208.
209. elif (body_aux['maxDataPoints']) == 100: #En el caso de que
   maxdatapoints sea 100
210. global auxi2 #Se invoca la variable
211. seleccion_aux = (auxi2%3)+1 #Se incrementa en 1
212. auxi2 += 1 #Se incrementa en 1
213.
214. elif (body_aux['maxDataPoints']) == 3: #En el caso de que
   maxdatapoints sea 3
215. auxi4 +=1 #Se incrementa en 1
216.
217. else: #En el caso de que no sea ninguno
218. global auxi3 #Se invoca la variable
219. seleccion_aux = (auxi3%3)+1 #Se incrementa en 1
220. auxi3 += 1 #Se incrementa en 1
221.
222. time_min_filter = time_min_aux #Guardamos el tiempo en una variable
   auxiliar
223.
224. try: #Se intenta
225.
226. with open(r"/data/prueba2/cuenta.txt", "r+") as f: #Abrimos el
   resultado
227. values = '[{"datapoints": [' #Creamos la variable con los valores
228. global coloumn2 #Cargamos variables globales dentro del with
229. coloumn2 = [] # Creamos la variable que será¡ rellenada
230. global n
231. n = 0 #Creamos un valor de control
232. data = f.readlines() #Leemos las lineas
233.
234. for line in data: #Para cada linea
235. if (n==1): #Si n es 1
236. if (body_aux['maxDataPoints']) == 100: #Si el maxdatapoints es 100
237. coloumn2.append(str(float(line.strip().split("|")[seleccion_aux])/5
   .95)) #Indezamos el valor
238. else: #Sino
239. coloumn2.append(line.strip().split("|")[seleccion_aux]) #Indexamos
   el valor
240.
241. n=1 #Ya nos hemos saltado la primera linea
242.
243. while True: #Para siempre
244. values = values + "[" + str(coloumn2[n-1][:].strip()) + ", " +
   str(time_min_filter*1000) + "], " #Indexamos el valor
245. time_min_filter += interval #Sumamos el tiempo indexado al
   intervalo
246. n+=1 #Aumentamos n
247.
248. if(time_min_filter>=(time_max_aux)): #Si nos hemos pasado de tiempo
249. break #Salimos del bucle
250.
```

```
251. values = values + "[" + str(coloumn2[n-2][:].strip()) + ", " +
   str(time_min_filter*1000) + "]]}]"  #Indexamos el último valor y
   cerramos
252. except:
253. hostPort = 9000
254.
255. #Aqui ya lo tenemos para enviar
256.
257. self.send_response(200) #Enviamos un OK
258. self.send_header("Content-type", "application/json; charset=UTF-8")
   #Indicamos el tipo
259. self.end_headers()
260. self.wfile.write(str.encode(values)) #Enviamos el JSON
261. myServer = HTTPServer((hostName, hostPort), MyServer) #Definimos el
   servidor
262. print(time.asctime(), "Server Starts - %s:%s" % (hostName,
   hostPort)) #Imprimimos petición por pantalla
263.
264. try:
265. myServer.serve_forever() #Se ejecuta siempre
266. except KeyboardInterrupt: #Mientras no se interrumpa
267. pass
268. reader.close() #Cerramos el reader
269. myServer.server_close() #Cerramos el servidor
270. print(time.asctime(), "Server Stops - %s:%s" % (hostName,
   hostPort)) #Imprimimos mensaje de cierre
```

Figura D-1 Servidor HTTP